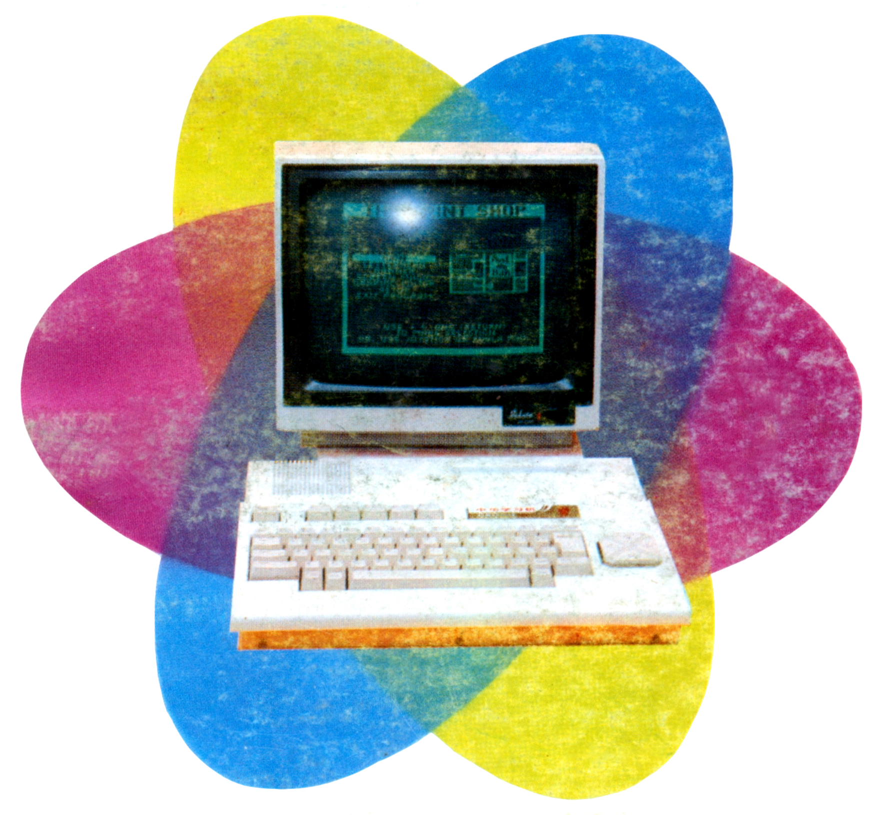
中华学习机实用大全⑤

汇编语言程序设计

韩仲清 主编



电子工业出版社

中华学习机实用大全⑤

汇编语言程序设计

韩仲清　主编

电子工业出版社

内容提要

本书从实用的角度出发，详细介绍了中华学习机CEC-I的指令系统和汇编语言。主要内容包括：计算机的工作原理，CEC-I硬件系统，6502指令系统，程序设计，源程序的编辑和运行以及CEC-I的监控系统。

本书的最大特点是内容充实、具体、实用、易学，最适宜于广大青少年、中小学生及其家长和计算机爱好者自学；也可以作为高等院校非计算机专业，培训班、函授班、职业学校、中专等学生计算机课程的教材，还可供从事计算机研究和应用的人员使用。

中华学习机实用大全⑤

汇编语言程序设计

韩仲清　主编

责任编辑　吴明卒

电子工业出版社出版（北京市万寿路）

电子工业出版社发行　各地新华书店经售

中国科学院印刷厂印刷

开本：787×1092毫米 1/32 印张：6.375 字数：147千字

1990年9月第1版　1991年7月第2次印刷

印数：10801—20900册　定价：2.50元

ISBN 7-5053-1052-6/TP·172

前　言

中华学习机以前所未有的速度进入寻常人家，成为人们工作、学习和生活的得力助手，尤其是在开发青少年的智力方面，已经显示出强大的威力。

为满足广大青少年、中小学生及其家长和计算机爱好者对中华学习机知识的渴求，我们组撰了这套《中华学习机实用大全》。该书内容丰富、具体、实用；把中华学习机的最新软件以及最实用、最急需的技术、技巧和方法毫无保留地介绍给读者，使初学者很快入门，入门者进一步提高；学到知识，掌握技术，增长才干，启迪智慧，得到力量，增强解决实际问题的能力。

《中华学习机实用大全》分为七册：

1．BASIC与LOGO语言

2．汉字处理与数据库技术

3．操作系统

4．FORTRAN与PASCAL语言

5．汇编语言程序设计

6．游戏与绘图

7．硬件维修与经验技巧

为便于阅读和使用，每册内容彼此均是独立的，读者可以从任何一本书开始阅读。但是，如果读者是计算机技术的初学者，那么最好按顺序阅读，当然，每本书中可以只选学自己感兴趣的那部分内容。

《中华学习机实用大全》在内容安排上，由浅入深，循序渐进。既考虑到初学者很快入门，又考虑到让入门者进一步提高，还考虑了应用者能够实用。书中有较多实例，读者可以边读、边学、边用、边想、边写（写自己的程序）。在结构安排上，既便于自学，又可以作为教材。在文字叙述上，力求浅显、通俗、易懂。在选材上，突出实用性技术。

本书是《中华学习机实用大全》的第五册，主要内容有：

中华学习机CEC-Ⅰ的汇编语言系统，各种寻址方式以及汇编语言的编程方法。这部分内容是其它中华学习机书籍中所没有的。如果读者希望了解CEC-Ⅰ的汇编语言，那么可以仔细阅读这本书。

欢迎读者对本书进行品评，指出疏漏和错误，我们将甚为感谢！

在编写本书的过程中，电子工业出版社和电子报社的编辑们给予了指导和帮助，提出了许多宝贵的修改意见；为调试和运行示例程序，成都三开元电脑部经理舒新生无偿地提供了CEC-Ⅰ中华学习机及其软件；张陞楷副教授审阅了全部书稿。在此一并表示感谢！

参加本书编写的有：许祖谦，丁正铨，韩仲清，王晓林，魏爱丽。全书由韩仲清统稿。

编　者

1989年11月14日于四川大学

目　　录

第一章　计算机的工作原理

1.1 引　　言

电子计算机是一种能高速地、准确地、自动地进行大量计算工作和信息处理的电子机器。由于它的工作方式和人脑思维过程有许多相似之处，所以人们又称它为“电脑”。

自1946年第一台电子计算机问世以来，已经经历了电子管、晶体管、集成电路、大规模集成电路和超大规模集成电路时代。由于大规模和超大规模集成电路的发展，1971年世界上诞生了第一个微处理器以后，以微处理器为核心的微型计算机便极其迅速地发展起来，差不多每两年就有一次重大进展，由4位机到8位机，到16位机。1981年以后又相继出现了32位微处理器及相应的微型计算机，目前正在向64位机发展。

较之大、中、小型计算机，微型计算机由于价格低，体积小，使用灵活方便，功能也足够强，因此，一经问世便获得飞速发展和广泛应用，目前已渗透到工业、农业、商业、国防、机关、家庭、娱乐游戏、家用电器等等方面，其应用遍及社会生产和生活的各个领域。

当前，世界正面临着一场新的技术革命，计算机技术的应用和普及程度已成为衡量一个国家现代化程度的重要标志

之一。所以，世界上许多国家已将计算机教育开始从高等学校转向中小学教育和家庭教育。由电子工业部组织研制的CEC-Ⅰ中华学习机是一种八位微型机，具有与APPLE-Ⅱe相当的功能，并有所增强，非常适合于家庭和中小学，是广大青少年学习计算机技术的有力工具。

1.2 电子计算机的硬件组成

电子计算机是作为一种计算工具而出现的。人用算盘算题时，需要用笔把参与运算的数据记在纸上，然后在大脑控制下，用算盘按一定规则进行运算，运算结果又用笔写在纸上。为了模仿人用算盘算题的过程，计算机也应具有完成上述功能的类似的组成部分：

① 能够进行数字运算的"算盘"——运算器；

② 能够记录和保存原始数据、运算步骤（程序）以及运算结果的"纸"——存储器；

③ 能够书写数据、程序和运算结果的"笔"——输入/输出设备；

④ 能够控制"算盘"、"纸"和"笔"协调工作的指挥者——控制器。

上述各部分之间的相互关系如图1.1所示。

运算器和控制器合在一起称为计算机的中央处理单元（常简称为CPU）。CPU与存储器，输入/输出接口（常简称为I/O口）一起组成计算机的主机；输入/输出设备统称为计算机的外围设备。

由图1.1中可以看到，在计算机中有两股信息在流动：

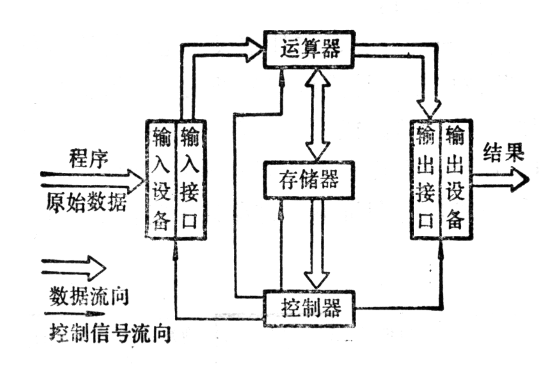


图1.1 电子计算机的硬件组成

一是数据流——各种原始数据，程序，现场信息等。这些要由输入设备输入到计算机并存于存储器中。在运算处理过程中，数据又从存储器读入运算器进行运算，运算结果再存入存储器或由输出设备输出。二是控制信息流——各种命令（或程序）。这些控制信息也以数据的形式输入到存储器中，运行时再由存储器读入控制器，由控制器经译码后变为各种控制信号。控制信号的作用是控制运算器的各种运算和处理，控制存储器的读和写，控制输入/输出设备的启动或停止等。

1.2.1 输入设备和输出设备

1．输入设备——输入数据和程序。常用的输入设备是键盘。输入设备和计算机之间的连接部分称为输入接口。

2．输出设备——输出计算机的处理结果。常用的输出设备是CRT显示器，打印机等。输出设备和计算机之间的连接

部分称为输出接口。

3．接口——计算机与外部设备之间的连接部分称为接口。如图1.2所示，计算机与外部设备交换信息时，都必须通过它们之间的接口，或者说，每一个外部设备都必须对应一个接口。比如，由键盘向计算机输入信息时，必须通过键盘接口；计算机向打印机输出打印结果时，必须通过打印机接口等等。

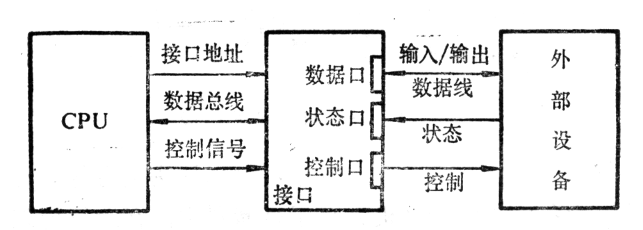


图1.2　CPU与外部设备之间的连接示意图

在最简单的情况下，每一个接口都必须要有一个数据缓冲寄存器（简称数据口）。在通常情况下，接口中除必须有数据口（暂存 输入/输出 数据）外，还有状态口（用来存放外设忙、闲的状态信息）和控制口（控制外设的启动和停止等等）。

计算机访问外设时，首先由CPU发出要访问的外设的地址和控制信号到该外设对应的接口，检查该外设接口中的外设是否准备好。若准备好，则CPU通过数据总线与接口中的数据口交换数据；接口再通过它与外设之间的 I/O 数据线与外设交换数据，这样就完成了计算机对外设的一次访问。若未准备好，则等到准备好后，再完成上述交换。所以，计算机对外设的控制，实际上是对其接口的控制，外设的地址也

是赋予其接口的。

1.2.2 存储器

存储器是计算机中的重要部件，其用途是存储数据和程序。计算机之所以能高速、自动地进行各种复杂的运算和控制，就是因为在解题之前已经把程序和数据事先存放在内存储器之中了。运行时，这些程序和数据又从内存快速提供给CPU进行分析和加工。

存储器分内存和外存。微型计算机的外存储器主要是磁盘；内存储器又分ROM和RAM。

ROM是只读存储器。在计算机运行过程中，只能从ROM读出数据，不能向它写入数据，所以，ROM通常用来存放计算机的系统程序和常数。RAM是读/写存储器，或称作随机存取存储器，在计算机运行过程中，可以对它随意进行数据的写入或读出。

内存储器，无论是ROM还是RAM都是分单元编号的。每一个单元有一个唯一确定的单元号码，称为地址码，正如每一间教室有一个唯一确定的教室号码一样。一个单元能存储一个字节或两个字节的二进制数。通常，每个字节规定为八个二进制位。

微型计算机中的内存储器是用大规模集成电路来实现的。它可以用触发器的两个不同的状态分别代表“0”和“1”，由此构成的存储器称为半导体静态存储器；也可以用集成电容上有、无电荷来分别代表“1”和“0”，由此构成的存储器称为半导体动态存储器。动态存储器必须周期性地自动将存储的数据重写，以维持电容器上的电荷，这种操作称为动态

存储器的刷新。CEC-Ⅰ机的RAM是使用的HM50464动态存储芯片，每片存储容量为64K×4位，两片并联使用，组成64K×8位的读/写存储器。

读/写存储器一般应由三部分组成，如图1.3所示。一部分

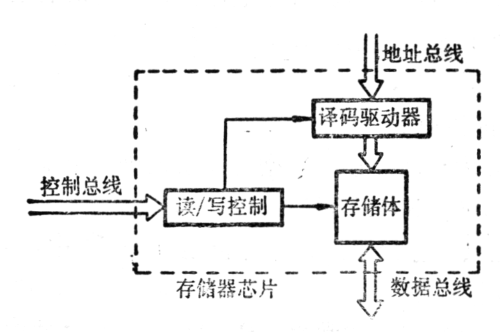


图1.3 存储器的原理结构框图

是能存储“0”和“1”的存储体，这是最基本的部分；第二是读/写控制部分，用来控制对该存储芯片的读/写；第三是译码驱动器，用来选择CPU要访问的某个单元。图1.4所示为HM50464动态RAM的逻辑引脚。

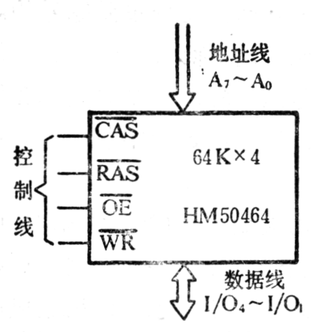


图1.4 HM50464动态RAM的逻辑引脚

由图1.3和图1.4可见，微机中的CPU通过控制总线、地址总线和数据总线来控制对存储器某一单元的读/写。

图1.4中的$\overline{OE}$是片选信号，$\overline{OE}$有效(低电平)时，表明CPU此

时可访问该存储器芯片。$\overline{WR}$是由CPU发来的读/写数据的控制信号，$\overline{WR}$为低电平时，表示CPU即将对存储器写入数据；$\overline{WR}$为高电平时，表示CPU即将从存储器中读出数据。$A_7 \sim A_0$是地址线，由它们分时输入地址总线的高八位和低八位地址。$\overline{CAS}$为列地址选通信号，$\overline{CAS}$有效时，由$A_7 \sim A_0$输入的地址为存储器的列地址；$\overline{RAS}$为行地址选通信号，$\overline{RAS}$有效时，由$A_7 \sim A_0$输入的地址为行地址。动态存储器都是由行地址和列地址共同选通要访问的存储单元。$I/O_4 \sim I/O_1$为数据输入/输出线。

存储芯片的存储容量定义为：总的单元数＊每单元的位数。HM50464芯片有八条地址线（$A_7 \sim A_0$），分时输入地址总线的高八位和低八位地址，分别作为芯片的行地址和列地址，所以它可以寻址的存储单元为$2^8 \times 2^8 = 2^{16} = 64 \times 2^{10} =$ 64K单元。芯片有四条数据线（$I/O_4 \sim I/O_1$），故每单元存储四位二进制数，所以每片HM50464芯片的存储容量为64K×4位。要用此芯片组装成每单元八位的存储器时，必须两片并联应用：一片存储低四位数据，另一片存储同一单元的高四位数据。

1.2.3 运算器

运算器（ALU）是进行算术运算（如加、减等）和逻辑运算（如与、或、异或等）的部件，其基本结构如图1.5所示。

ALU是运算器的核心部分，称为算术逻辑运算单元。ALU的两个操作数一个来自累加寄存器，另一个来自暂存寄存器；运算结果一方面通过CPU内部总线送往累加寄存器，同时将运算结果的标志送往标志状态寄存器。

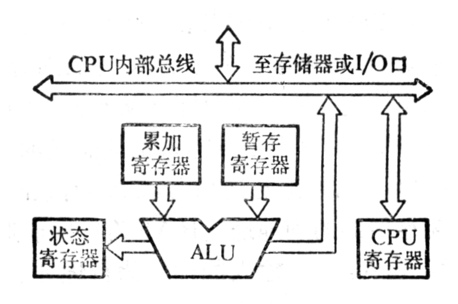


图1.5　运算器原理简图

1.2.4　控制器

控制器用于指挥和协调上述各部分的工作，它由指令寄存器（IR），指令译码器（ID）和操作控制部件组成，如图1.6所示。

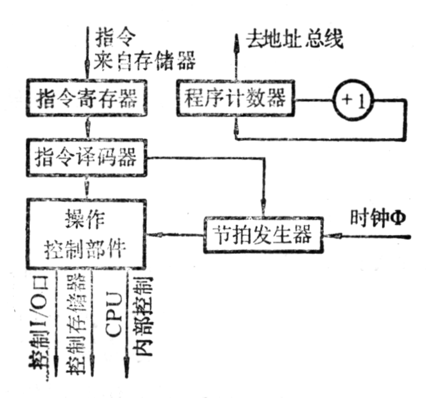


图1.6　控制器结构简图

运行程序时，CPU从存储器取出的指令首先送入指令寄

存器，然后送入指令译码器，经过译码分析，送入操作控制部件，产生一系列控制信号，并在节拍发生器控制下，按一定时间顺序送到计算机的不同部件，控制I/O接口、存储器和CPU内部。时钟的作用是产生定时信号。节拍发生器在时钟驱动下按一定周期产生执行指令所需的节拍电位和节拍脉冲，控制计算机按一定时序有条不紊地正常执行。

为了指明下一条即将执行的指令在哪一个单元，CPU设置了一个指令地址计数器PC。计算机在运行程序时，首先把PC中内容（即下一条即将执行的指令存放的地址）送到地址总线，选通所指的地址单元，然后从该单元读出指令。当PC的内容送到地址总线并稳定后，PC内容将自动加1，形成下一条指令的地址。由于PC的内容反映了程序执行的顺序，所以常称PC为程序计数器。

1.3 电子计算机的系统组成

一个完整的计算机系统应包括硬件系统和软件系统两大部分，其主要组成如图1.7。

一个具体的计算机系统所包含的硬件和软件数量是各不相同的，这与计算机的规模、特性、应用场合等密切相关，依具体情况而定。

1.4 微型计算机的硬件系统结构

由于大规模集成电路工艺的发展，计算机的中央处理单元CPU可以缩微在一片或几片大规模集成电路芯片上。通

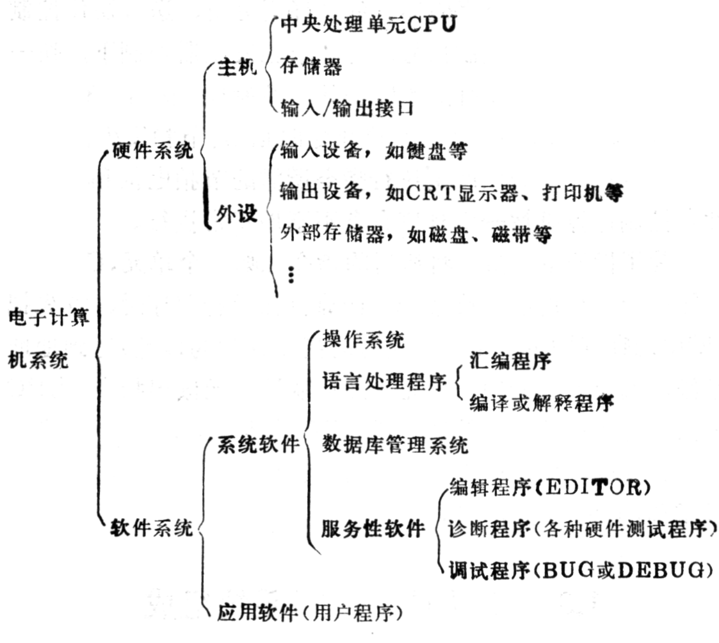


图1.7 电子计算机的系统组成

常，把这种用大规模集成电路来实现的CPU称为微处理器（常简称为MPU)。以微处理器为核心，再加上由大规模集成电路实现的存储器,输入/输出接口等组成的计算机称为微型计算机。微型计算机再配上所需的外部设备等就构成微型计算机的硬件系统；硬件系统加上必须的系统软件和应用软件就构成一个微型计算机系统。

目前，大多数微型计算机都是采用一束公共的信息传输线——系统总线来实现CPU、存储器、输入/输出接口之间的信息交换；或者说,CPU通过系统总线实现与存储器、输入/输出接口之间的相互连接。这种系统结构常称为单总线结

构，如图1.8所示。

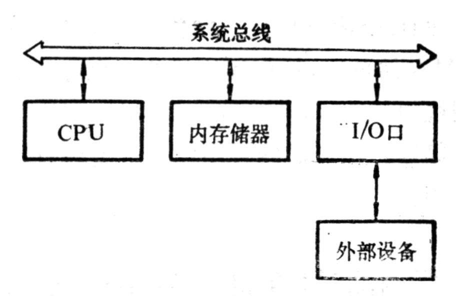


图1.8　单总线系统结构

根据系统总线上所传输信息的性质，系统总线又分为地址总线、数据总线和控制总线三大类。所以，有的书上又称三总线结构。典型的微型计算机硬件的系统结构如图1.9所示。微处理器通过地址总线指定它要访问的内存储器单元的

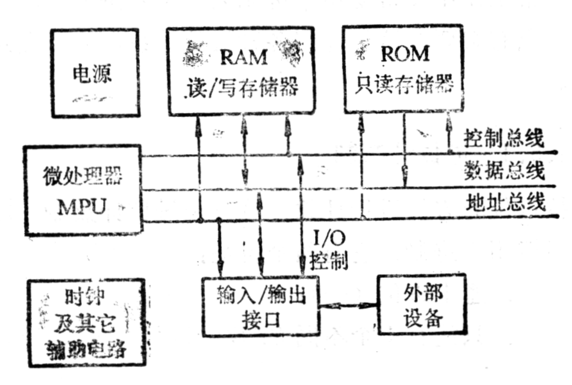


图1.9　微型计算机硬件的系统结构框图

地址码或输入/输出口的地址码；通过数据总线与存储器、输入/输出接口交换数据；通过控制总线发出控制信息，控制

存储器和输入/输出接口的工作。八位机一般有16根地址线，可寻址$2^{16}=64K$个内存单元。16位机一般有20根地址线，可寻址$2^{20}=1M$存储单元。数据线的根数（也称为数据线宽度）决定于微型机的位数，比如8位机有8根数据线，16位机有16根数据线，等等。控制总线对不同的机器略有不同，大约有几根到十几根。

1.5 数和符号在计算机中的表示方法

从1.2节中我们已经知道，无论数据还是程序都以数字的形式存放于内存储器中。内存储器在本质上是一组触发器，每个触发器只有两个可能的状态，我们用它分别表示二进制数的“0”和“1”，这就是计算机总是采用二进制的原因。而我们日常工作中又常使用十进制数，十六进制数，八进制数等，所以有必要在这里介绍各种进位计数制的特点及其相互转换。

1.5.1 进位计数制

按进位的方法进行计数，这就是进位计数制。比如日常算数习惯用十进制，它有两个特征：

第一，逢十进位（一）；

第二，只能有十个不同的数字符号0,1,2,3,4,5,6,7,8,9。

因此，任一个十进制数都可以表示成它的数字符号与权相乘后的组合，即分解成10的方次和的形式。比如：

$$143.52=1\times10^{2}+4\times10^{1}+3\times10^{0}+5\times10^{-1}+2\times10^{-2}$$

一般情况下，任一个十进制数A可表示为：

$$A = A_{n-1}\times 10^{n-1} + A_{n-2}\times 10^{n-2} + \cdots + A_1\times 10^1 + A_0\times 10^0 + A_{-1}\times 10^{-1} + A_{-2}\times 10^{-2} + \cdots + A_{-m}\times 10^{-m}$$

$$= \sum_{i=n-1}^{-m} A_i \times 10^i \qquad (1)$$

式中，n是小数点左边的总位数；m是小数点右边的总位数；A_i是第i位的数字，可为0,1,2,3,4,5,6,7,8,9中任一个；10^i表示第i位的权，即当第i位数字为1，其余各位为0时第i位的大小。比如十进制数$234 = 2\times 10^2 + 3\times 10^1 + 4\times 10^0$。式中$10^2$，$10^1$，$10^0$分别是十进制数234的第2位，第1位和第0位的权。第2位的权为10^2，故其大小为200；第1位的权为10^1，故其大小为30；第0位的大小为$4\times 10^0 = 4$。权的概念在计算技术中十分重要。

又如二进制数，它也有两个特征：

第一，逢2进位（一）；

第二，只有两个数字符号0，1。

同样，任一个二进制数都可表示成它的数字符号与其权相乘后的组合，即分解成2的方次和的形式。如二进制数1001.01可表示为：

$$1001.01 = 1\times 2^3 + 0\times 2^2 + 0\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}$$

对二进制数B可定义为：

$$B = B_{n-1}\times 2^{n-1} + B_{n-2}\times 2^{n-2} + \cdots + B_0\times 2^0 + B_{-1}\times 2^{-1} + \cdots + B_{-m}\times 2^{-m}$$

$$= \sum_{i=n-1}^{-m} B_i 2^i \qquad (2)$$

式中B_i是第i位的数字，它只能取0或1；2^i是第i位的权，它确定了第i位数字的大小；m和n的含义同前。

综上所述，对于任一进制数N可表示为：

$$N = \sum_{i=n-1}^{-m} N_i J^i \tag{3}$$

式中：

① J为进制数的基数，逢J进位。比如二进制中$J=2$，逢2进位；十进制中$J=10$，逢十进位；16进制中，$J=16$，逢16进位，等等。

② J^i为数N在第i位的权。如：

$$(2468)_{10} = 2\times \underline{10^3} + 4\times 10^2 + 6\times 10^1 + 8\times \underline{10^0}$$

第3位的权 … 第0位的权

$$(10010)_2 = 1\times \underline{2^4} + 0\times 2^3 + 0\times \underline{2^2} + 1\times 2^1 + 0\times \underline{2^0}.$$

第4位的权 … 第2位的权 … 第0位的权

③ N_i为第i位的数字，它只能为$0\sim(J-1)$之间的任一整数。比如当：

$J=2$时，　$N_i=0,1$

$J=10$时，$N_i=0,1,2,3,4,5,6,7,8,9$

$J=16$时，$N_i=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15$

对16进制数大于9的数字符号用字母表示，所以16进制数的数字符号通常写作：

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

④ n是小数点左边的总位数。

⑤ m是小数点右边的总位数。

1.5.2 不同进位计数制之间的转换

1. 从二进制数转换为十进制数

根据二进制数的定义，将二进制数按权展开相加即得到对应的十进制数。比如：

$$
\begin{aligned}
(111.101)_2 &= 1\times 2^2 + 1\times 2^1 + 1\times 2^0 \\
&\quad + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} \\
&= 4+2+1+0.5+0.125 \\
&= (7.625)_{10}
\end{aligned}
$$

或写成111.101B = 7.625D。B是二进制数标志，D是十进制数标志。标志B和D通常约定可以不写。

2. 十进制数转换为二进制数

① 十进制整数转换为二进制整数的方法是：将十进制整数不断除以2，直到商为0，反序取余数，即得对应的二进制整数。比如：

除法	余数
2 | 167	
2 | 83	……1
2 | 41	……1
2 | 20	……1
2 | 10	……0
2 | 5	……0
2 | 2	……1
2 | 1	……0
0	……1

故 $(167)_{10} = (10100111)_2$

② 十进制小数转换为二进制小数的方法是：将十进制小

数不断乘以2，顺序取整数，直到小数部分为0。比如

```
   0.8125
 ×      2           整数
 ─────────
   1.6250     ···········1

   0.625
 ×     2
 ─────────
   1.250      ···········1

   0.25
 ×    2
 ─────────
   0.50       ···········0

   0.50
 ×   2
 ─────────
   1.00       ···········1
```

故 $(0.8125)_{10}=(0.1101)_2$

有时会遇到无限乘下去都得不到小数部分为0的情况，这时可根据问题要求的精度取足够的二进制位即可。

③ 既含整数又含小数的任一十进制数转换为二进制数的方法是：将十进制数的整数部分和小数部分分别转换为二进制数的整数和小数，然后用小数点把整数部分同小数部分连接起来即可。比如：

$$(167.8125)_{10}=(10100111.1101)_2$$

3. 八进制数与十进制数之间的转换

与二-十进制数之间的转换相似。

八进制数→十进制数：将八进制数按权展开相加即得对应的十进制数。

十进制整数→八进制整数：将十进制整数不断除以8，直

至商为0，反序取余数即得对应的八进制整数。

十进制小数→八进制小数：将十进制小数不断乘以8，顺序取整数，直至小数部分为0（或取足够的位数），即得八进制小数。

4．二进制数和八进制数之间的转换

因为$2^3=8$，所以一位八进制数相当于三位二进制数，它们彼此完全对应。

① 二进制数转换为八进制数

转换方法是：从小数点开始，分别往左和往右数，每三位一组，不足三位的用0补足三位，然后把每三位二进制数用相应的八进制数表示即可，如1110111.1011：

$$(\underline{001}\ \underline{110}\ \underline{111}.\underline{101}\ \underline{100})_2=(167.54)_8$$

（前面添0补足三位）↓1 ↓6 ↓7 . ↓5 ↓4（后面添0补足三位）

或写作1110111.1011B＝167.54Q，Q是通常使用的八进制标志，不能省略。在CEC-Ⅰ机中，八进制数标志是@，而且是放在数的前面。如167.54Q在CEC-Ⅰ机中应表示为@167.54。

② 八进制数转换为二进制数

转换方法是：把每位八进制数转换为相应的三位二进制数即成。比如：

$$(2\ 5\ 4\ .\ 1\ 2)_8=(10101100.00101)_2$$

↓010 ↓101 ↓100 . ↓001 ↓010

5．二进制数和十六进制数之间的转换

因为$2^4=16$，故可以用一位16进制数表示四位二进制数。这样，一字节（八位）二进制数就可以用两位16进制数非常方便地表示出来，所以16进制数在计算机的输入/输出中

十分重要，比如，CRT屏幕显示和打印机都是输出的16进制数或用16进制表示的字符的ASCII码。

① 十六进制数转换为二进制数

转换方法是：把每位十六进制数用相应的四位二进制数代替。比如：

$$
\begin{array}{ccccc}
(3 & A & B & . & 1)_{16} = (1110101011.0001)_2 \\
\downarrow & \downarrow & \downarrow & & \downarrow \\
0011 & 1010 & 1011 & . & 0001
\end{array}
$$

通常是用H作为十六进制数的标志，故 $(3AB.1)_{16}$ 可写为3AB.1H。但在CEC-Ⅰ机中，却用＄置于数前来表示16进制数。比如，3AB.1H在CEC-Ⅰ机中应表示为＄3AB.1，希望读者特别注意。

② 二进制数转换为十六进制数

方法是：从二进制数的小数点开始，分别往左和往右数，每四位一组，不足四位时用0补足四位，然后，把每四位一组的二进制数用相应的十六进制数代替即可。比如，101101010.10111_2

$$
\begin{array}{ccccccc}
 & (\underline{0001} & \underline{0110} & \underline{1010} & . & \underline{1011} & \underline{1000})_2 = (16A.B8)_{16} & \\
\left(\begin{matrix}\text{用0补足}\\\text{四位}\end{matrix}\right) & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \left(\begin{matrix}\text{用0补足}\\\text{四位}\end{matrix}\right) \\
 & 1 & 6 & A & . & B & 8 &
\end{array}
$$

或写成101101010.10111B＝＄16A.B8

1.5.3 带符号数在机器中的表示方法

一个实际的数，如＋34.25，它有三个重要特征：

① 有效数字，如＋34.25的有效数字是3425。

② 小数点位置

比如34.25可用小数点位置E＝－2表示，此时：34.25＝

3425×10⁻²；也可用E＝＋2表示，此时34.25＝0.3425×10⁺²。这样，一个含小数的任意数，都可以用两个带符号整数——有效数字及小数点位置来表示。

③ 正负符号：+，-

在计算机内部，有效数字（比如3425）可以通过上述进制数的转换方法转换为二进制数，参与运算。但小数位置如何表示？数的正负符号如何表示？下面我们只讨论带符号整数的表示方法。小数的表示方法请参见有关书籍。

在计算机中，处理正负符号的基本思想是把符号数字化。其中最常用的是补码表示法和原码表示法。

1. 原码表示法

原码表示法又称为符号大小表示法。它是把码的最高位作为数的符号位，其余位表示数的绝对值大小；最高位为0代表正数，最高位为1代表负数，这就是原码表示法。比如：

$$[+127]_{原}=\underline{0}\ \underline{1111111}$$

符号位 数值位(127)

$$[-127]_{原}=\underline{1}\ \underline{1111111}$$

符号位 数值位(127)

原码表示法的数学定义为：

$$[X]_{原}=\begin{cases}X & 0\leqslant X\leqslant 2^{n-1}-1\\ 2^{n-1}+|X| & -(2^{n-1}-1)\leqslant X\leqslant 0\end{cases} \quad (4)$$

式中n为二进制数的位数。可见当$X\geqslant 0$时，X的原码为X自身；$X\leqslant 0$时，则在X的绝对值前加上1，表示负号。以后，我们把带符号数X称为真值，$[X]_{原}$表示X的原码。

由式(4)可知，原码所能表示的真值范围是$-(2^{n-1}-1)$

$\sim +(2^{n-1}-1)$；当$n=8$时，原码所能表示的真值范围是$-127\sim+127$。这是因为在原码表示法中，用了最高位表示符号，剩下只有$n-1$位用于表示数值大小。

如何由真值求原码？

首先，将X转换成带符号二进制数。然后由式（4）可知，如果

$X\geqslant 0$，则$[X]_{原}=X$，即最高位置0，其余位不变。

$X\leqslant 0$，则$[X]_{原}=2^{n-1}+|X|$，即去掉负号，最高位置1，其余位不变。

比如，当$n=8$时，若$X=-9$，则：

$X=-0001001$　　（转换成带符号二进制数）

$[X]_{原}=\underline{1}\,\underline{0001001}$（去掉负号，最高位置1，其余位不变）

置1　　不变

如何由原码求真值？

如果已知$[X]_{原}$的最高位是0，表示X为正数，此时真值$X=[X]_{原}$；

如果已知$[X]_{原}$的最高位是1，则表示X是负数，此时真值$X=-([X]_{原}-2^{n-1})$，即将$[X]_{原}$的最高位1去掉（减2^{n-1}）变成0后，前面再添上负号，便得真值X。比如：

$[X]_{原}=10000001$，则$X=-([X]_{原}-2^{n-1})=-00000001$

$[X]_{原}=10000000$，则$X=-([X]_{原}-2^{n-1})=-00000000$

$[X]_{原}=00000000$，则$X=[X]_{原}=00000000$

可见，原码可以分别表示出$+0$和-0。

原码表示法简单易懂，易记。但要注意的是：$[X]_{原}+[Y]_{原}\neq[X+Y]_{原}$。比如当$n=4$时：

$$[-3]_{原}=1011，\quad [+1]_{原}=0001$$

$$[-3]_{原}+[+1]_{原}=1011+0001=1\ 100=[-4]_{原}$$

符号　数值

可见，$[-3]_{原}+[+1]_{原}=[-4]_{原}$，而不是$[-2]_{原}$，这表明在原码表示法中，码值的运算与真值的运算不能一一对应。所以，原码表示法在微机中不能得到广泛的应用，于是人们又提出了补码表示法。

2．补码表示法

一个数的补码在数学上可定义为：

$$[X]_{补}=\begin{cases} X & 0\leqslant X<\dfrac{M}{2} \\ M+X=M-|X| & -\dfrac{M}{2}\leqslant X<0 \end{cases} \quad (5)$$

式中M称为模。模是指系统的最大量程。比如时钟面值的最大量程是12，超过12产生进位被丢掉；一字节存储单元能存八位二进制数，其最大量程为255，超过255产生进位。

由式（5）可知，在补码表示法中，真值X的符号是寄于码值的大小之中。当$[X]_{补}<\dfrac{M}{2}$时，$X\geqslant0$；当$[X]_{补}\geqslant\dfrac{M}{2}$时，$X<0$，这是与原码表示法的本质区别。在原码表示法中，原码的最高位是专门的符号位，其余位是数值位；在补码表示法中，由补码码值的大小来判断真值的正、负。对于公式（5）的定义有：

当$M=2^n$时，称$[X]_{补}$为2的补码，即常说的补码；

当$M=2^n-1$时，称为1的补码，即常说的2的反码；

当$M=10^n$时，称为10的补码；

当$M=10^n-1$时，称为9的补码，即10的反码。

对于2的补码，由式（5）可得：

$$[X]_{补}=\begin{cases}X & 0\leqslant X\leqslant 2^{n-1}-1\\ 2^n+X & -2^{n-1}\leqslant X<0\end{cases} \quad (6)$$

可见，当$n=8$时，补码所能表示的真值范围是$[-2^{8-1}, +(2^{8-1}-1)]$ 即$[-128, +127]$，而且补码只能表示$+0$，不能表示-0。

如何由真值求补码？（下面我们只讨论二进制补码。）

按照式（6）的定义可知：

当$X\geqslant 0$时，$[X]_{补}=X$；

当$X<0$时，$[X]_{补}=2^n+X=2^n-|X|$。

若我们假定$\overline{|X|}$为$|X|$各位取反，则有

$|X|+\overline{|X|}=2^n-1$

故 $[X]_{补}=2^n-|X|$

$$=|X|+\overline{|X|}+1-|X|$$

$$=\overline{|X|}+1 \quad (7)$$

所以，当X为负数时，$[X]_{补}$等于X的绝对值各位取反后末位加1。

由X求$[X]_{补}$的具体办法是：

首先将X转换为n位带符号二进制数，比如：

$+127=+01111111$； $-127=-01111111$

①如果X是正数，则$[X]_{补}=X$（去掉正号），如

$$[+127]_{补}=[+01111111]_{补}=01111111$$

② 如果X是负数，则$[X]_{补}$等于X的二进制数各位取反后末位加1，去掉负号。如：

$$[-127]_{补}=[-01111111]_{补}=10000001$$

由此可见，在二进制补码表示法中，仍然可用码的最高位表示符号，即最高位为0代表正数；最高位为1代表负数。但在补码表示法中，码的最高位不仅仅表示符号，它同时还要参与求数值大小的运算，这是与原码表示法在本质上不同的地方。

③ 求$[-X]$的补码

证明：设 $[X]_{补} = X$；$\overline{[X]}_{补}$ 为$[X]_{补}$的各位取反。

则 $[-X]_{补} = 2^n + (-X) = 2^n - X$

$$= 2^n - [X]_{补}$$

又因为 $[X]_{补} + \overline{[X]}_{补} = 2^n - 1$

故 $2^n = [X]_{补} + \overline{[X]}_{补} + 1$

故 $[-X]_{补} = [X]_{补} + \overline{[X]}_{补} + 1 - [X]_{补}$

$$= \overline{[X]}_{补} + 1$$

可见，$[-X]$的补码等于X的补码各位取反，末位加1。

如何由补码求真值？

如果$[X]_{补}$的最高位是0，则X为正数，$X = [X]_{补}$。比如，当$M = 2^4$时，若$[X]_{补} = 0101$，最高位为0，故X应为正数，于是$X = [X]_{补} = 0101$。

如果$[X]_{补}$的最高位是1，则X为负数，此时由式(6)可得：

$$X = [X]_{补} - M = [X]_{补} - 2^n$$

因为 $[X]_{补} + \overline{[X]}_{补} = 2^n - 1$

故 $X = [X]_{补} - ([X]_{补} + \overline{[X]}_{补} + 1)$

$$= -(\overline{[X]}_{补} + 1) \qquad (8)$$

式(8)表示，当$[X]_{补}$最高位为1时，真值X等于$[X]_{补}$各

位取反，末位加1，前面再冠以负号。比如，当$M=2^4$时，若：

$$[X]_{补}=1101$$

$$\text{则}\quad X=-(\overline{[X]_{补}}+1)$$

$$=-0011$$

在补码表示法中，码值的运算与真值的运算一一对应，即$[X]_{补}+[Y]_{补}=[X+Y]_{补}$。比如，对于$M=2^4$，若

$$[-3]_{补}=1101，[+1]_{补}=0001$$

$$\text{则}\quad [-3]_{补}+[+1]_{补}=1101+0001=1110=[-2]_{补}$$

$$\text{故}\quad [-3]_{补}+[+1]_{补}=[-3+1]_{补}=[-2]_{补}$$

即和的补码等于补码的和。在一般情况下，补码这一性质可表示为：

$$([X]_{补}+[Y]_{补})\text{Mod}\,M=[X+Y]_{补} \qquad (9)$$

式中Mod M是丢掉进位位的意思。亦就是说，两个数和的补码等于这两个数补码之和丢掉进位位。比如，对于$M=2^8$，若$X=-1$，$Y=4$，它们之和$X+Y=3$，这两个带符号数求和的运算可以通过对它们的补码求和的运算来代替。比如：

$$X=-00000001, \qquad Y=00000100$$

$$[X]_{补}=11111111, \qquad [Y]_{补}=00000100$$

$$\begin{array}{rl} & [X]_{补}=11111111 \\ + & [Y]_{补}=00000100 \\ \hline & \underset{\text{丢掉进位}}{\overset{1}{\uparrow}} \quad \underbrace{00000011}_{\text{结果}}=[+3]_{补} \end{array}$$

可见，$[-1]_{补}+[+4]_{补}$丢掉进位后刚好等于$[+3]_{补}$。这样，

码值运算与真值的运算一一对应，对真值的运算就可以通过对其补码的运算来实现。所以补码表示法在微机中得到了广泛的应用，除非特别声明，微机中一般都是采用补码表示法。

3．进位与溢出

从前面的讨论中已经知道，在微机中一般都是进行的补码运算。而在补码运算中将涉及到进位和溢出这两个不同的概念。

进位是指码值超过了n位二进制数所能表示的范围（即码值$\geq M$）。比如，当$M=2^8$时，如果运算结果码值大于255，就会产生进位。在补码运算中，丢掉进位后的结果正确，运算器正是这样处理的。正如时钟面值的模$M=12$，当时间超过12点时，丢掉进位又从0点、1点开始计时一样，时钟面值上不出现13点，14点等等，但结果仍然正确。

溢出是指带符号数真值的大小超过了码值所能表示的范围。比如$M=2^8$时，补码所能代表的带符号数的真值范围是$-128\sim+127$，如果真值超过了这个范围就产生溢出，结果就会出错。

例1．设$M=2^8$，$X=+65$，$Y=+64$，则：

$$
\begin{array}{rl}
 & [X]_{补}=01000001 \\
+ & [Y]_{补}=01000000 \\
\hline
 & 10000001=[-127]_{补}
\end{array}
$$

$65+64$应等于$+129$，但补码运算结果表明，真值等于-127，这说明运算结果出错，其原因就是因为$+129$超过了$M=2^8$所能表示的范围$-128\sim+127$，产生了溢出所致。

例2．设$M=2^8$，$X=-65$，$Y=-64$，$X+Y$的结果应

等于－129，但计算机中进行补码运算的结果为：

$$
\begin{array}{rl}
 & [X]_{补} = 10111111 \\
+ & [Y]_{补} = 11000000 \\
\hline
 & 1\quad 01111111 = [+127]_{补}
\end{array}
$$

↑
进位

运算结果为＋127，显然错误。这也是因为－129超过了 $M=2^8$ 所能表示的真值范围，产生了溢出所致。

当运算结果产生溢出时，应停止程序运行，进行处理。

例3．设 $M=2^8, X=-64, Y=-63$，则 $X+Y=-127$，在计算机中进行补码运算的结果为：

$$
\begin{array}{rl}
 & [X]_{补} = 11000000 \\
+ & [Y]_{补} = 11000001 \\
\hline
 & 1\quad 10000001 = [-127]_{补}
\end{array}
$$

↑
进位

可见运算结果正确。因为－127没有超出八位补码所能表示的真值范围，故不会产生溢出，丢掉进位后结果仍然正确。

例4．设 $M=2^8$，$X=64$，$Y=63$，则 $X+Y=+127$，在计算机中进行的补码运算结果为：

$$
\begin{array}{rl}
 & [X]_{补} = 01000000 \\
+ & [Y]_{补} = 00111111 \\
\hline
 & 01111111 = [+127]_{补}
\end{array}
$$

运算结果正确。因为＋127在八位补码所能表示的真值范围内，没有溢出。

从上面四个例子不难看出，判断溢出最方便也最常用的

方法是符号位判断法。方法是：

设X_0为$[X]_{补}$的符号位，Y_0为$[Y]_{补}$的符号位，Z_0为运算结果的符号位。如果：

① $X_0=0$，$Y_0=0$，$Z_0=0$，则无溢出，如例4；

② $X_0=1$，$Y_0=1$，$Z_0=1$，则无溢出，如例3；

③ $X_0=0$，$Y_0=0$，$Z_0=1$，则有溢出，如例1；

④ $X_0=1$，$Y_0=1$，$Z_0=0$，则有溢出，如例2。

以上判断方法可用下面的逻辑式表示，逻辑变量

$$V=X_0Y_0\overline{Z}_0+\overline{X}_0\overline{Y}_0Z_0=\begin{cases}0 & \text{无溢出}\\1 & \text{有溢出}\end{cases} \tag{10}$$

在微型计算机的CPU中，常将V作为它的标志寄存器中的一位，用于记载和考查运算结果是否溢出。如果V位为1，有溢出；如果V位为0，无溢出，故V位称为溢出位。我们可以通过考查V位的状态来判断此次运算结果是否产生了溢出。

由以上例子还可以看出：

① 有进位时不一定有溢出，如例3；有溢出时不一定有进位，如例1。进位是对码值而言，溢出是对真值而言，它们是完全独立的两个概念，务必要弄清楚。

② 溢出发生时，结果错误，此时操作人员应停止程序运行，进行处理；有进位发生而无溢出时，不影响运算结果的正确性。

1.5.4 二-十进制编码

在计算机中的运算器只能进行二进制数运算，但有些问题要求我们在计算机中直接进行十进制数的运算，这时怎么

办呢？首先要对十进制数进行二进制编码，这种编码称为二-十进制数编码。编码后就可借助计算机中的二进制数运算器进行十进制数运算，但运算过程中必须进行适当的调整（称为十进制调整）。在有的微型机CPU中，设有专门的十进制调整指令（如Z-80CPU），通过执行该指令就实现了对十进制数运算的调整。CEC-Ⅰ机的CPU，是在它的标志寄存器中设了一位十进制调整位D，若置该位为1，则以下进行的算术运算即为十进制运算。详细情况请见第三章。

常用的二-十进制编码如表1.1所示，它们都是用四位二进制数表示一位十进制数，但各自编码的规律不同。

表1.1 常用二-十进制编码

十进制数	8421BCD码	2421BCD码	余3代码	循环码
0	0000	0000	0011	0000
1	0001	0001	0100	0001
2	0010	0010	0101	0011
3	0011	0011	0110	0010
4	0100	0100	0111	0110
5	0101	1011	1000	0111
6	0110	1100	1001	0101
7	0111	1101	1010	0100
8	1000	1110	1011	1100
9	1001	1111	100	1000

① 8421BCD码是将四位码从高位到低位的权分别设为8,4,2,1。这样，8421码就与一位十进制数通常所对应的二进

制数一致，因此简单、易记。比如，9的8421码为1001，它各位的权分别为：

$$8421\text{BCD码}\ \underset{\substack{\downarrow\ \downarrow\ \downarrow\ \downarrow\\ 8\ 4\ 2\ 1}}{1\ 0\ 0\ 1}\xrightarrow{\text{对应的十进制数}}1\times8+0\times4+0\times2+1\times1=9$$

各位的权 8 4 2 1

② 2421BCD码的四位码从高位到低位的权分别是2，4，2，1。比如9的2421BCD码为1111，它各位的权为

$$2421\text{BCD码}\ \underset{\substack{\downarrow\ \downarrow\ \downarrow\ \downarrow\\ 2\ 4\ 2\ 1}}{1\ 1\ 1\ 1}\xrightarrow{\text{对应的十进制数}}1\times2+1\times4+1\times2+1\times1=9$$

各位的权 2 4 2 1

③ 余3代码是指它与对应的8421BCD码的差为3。

④ 循环码的每两个相邻码之间只差一位，因此可靠性高，常用于通信中。

最常用的二-十进制代码是8421BCD码，若无特别说明，通常所说的BCD码均为8421BCD码。2421码和余3代码求十进制数的反码方便，只需将各位取反即成。

需要注意的是：用8421BCD码代表十进制数时，虽然0，1，2，…9这十个数字的代码与它们对应的二进制数相同，但BCD码毕竟只是十进制数的二进制代码，并不是等效的实际的二进制数。比如：

十进制数	对应的8421 BCD码	等效的二进制数
1	0001	0001
10	0001 0000	1010
100	0001 0000 0000	01100100
255	0010 0101 0101	11111111

前面已经讨论了带符号数和十进制数在计算机中的表示

方法。下面我们将机器中的一字节二进制数（n = 8）看作不同的代码时所对应的真值列于表1.2中。

表1.2 机器中不同代码对应的真值

机器中的二进制数 n = 8	看作不同代码时对应的真值			
	无符号数	原 码	补 码	反 码
0000 0000	0	+0	+0	+0
0000 0001	1	+1	+1	+1
0000 0010	2	+2	+2	+2
⋮	⋮	⋮	⋮	⋮
0111 1111	127	+127	+127	+127
1000 0000	128	−0	−128	−127
1000 0001	129	−1	−127	−126
⋮	⋮	⋮	⋮	⋮
1111 1110	254	−126	−2	−1
1111 1111	255	−127	−1	−0

1.6 ASCII码

ASII码是美国信息交换标准码的缩写，是微小型计算机中常用字符、字母、数字的二进制代码。绝大多数微型机的键盘输入，CRT终端显示以及点阵式打印机的输出，都采用七位ASCII码（$A_6 \sim A_0$）。因为$2^7 = 128$，故ASCII码能表示128个字符、字母、数字和符号等。如表1.3所示。通常将最高位A_7位作为奇偶校验位，不用A_7位时将该位置0。但在

表1.3 ASCII字符编码表

列		0(3)	1(3)	2(3)	3	4	5	6	7(3)
行	位 654→ ↓3210	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	”	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	’	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	〔	k	{
C	1100	FF	FS	,	<	L	＼	l	|
D	1101	CR	GS	—	=	M	〕	m	}
E	1110	SO	RS	·	>	N	Ω(1)	n	~
F	1111	SI	US	/	?	O	—(2)	o	DEL

注：(1) 取决于使用这种代码的机器，它的符号可以是弯曲符号，向上箭头，或（—）标记。

(2) 取决于使用这种代码的机器，它的符号可以是在下面画线，向下箭头，或心形。

(3) 是第0、1、2和7列特殊控制功能的解释。

CEC-Ⅰ机中使用的ASCII码，其最高位是置1，请注意这

点。比如字母“A”的标准ASCII码是“41”(十六进制表示),回车符“CR”的标准ASCII码是“0D”(十六进制表示),但在中华学习机中,“A”的ASCII码是“C1”(十六进制表示);“CR”的ASCII码是“8D”(十六进制表示)。

习 题

1. 计算机的硬件系统包括几大部分?各部分的功能是什么?

2. 为什么目前的计算机都采用二进制?

3. 将下列十进制数转换为等效的二进制数。

$51=(\qquad)_2$; $0.375=(\qquad)_2$

$131.375=(\qquad)_2$; $25/32=(\qquad)_2$

4. 把下列二进制数转换为等效的十进制数。

$(11010)_2=(\qquad)_{10}$; $(11011.0101)_2=(\qquad)_{10}$

$(1001.01011)_2=(\qquad)_{10}$; $(111011.10111)_2=(\qquad)_{10}$

5. 将第4题中各二进制数转换为等效的八进制数和十六进制数。

6. 将下列十进制数转换为等效的十六进制数。

$(1863)_{10}=(\qquad)_{16}$; $(563)_{10}=(\qquad)_{16}$

$(562.345)_{10}=(\qquad)_{16}$; $(134.5)_{10}=(\qquad)_{16}$

7. 写出下列十进制数X的①八位原码;②$M=2^8$的补码;③$M=2^8-1$的补码

$X=10$ $X=127$

$X=64$ $X=-64$

$X=-127$

8. 简述如何从原码判断真值的符号。

第二章　CEC-Ⅰ硬件系统

2.1　CEC-Ⅰ的主机组成

CEC-Ⅰ中华学习机与其它微机一样，硬件系统由五大部分组成。在CEC-Ⅰ机中，除外围设备外，其余部分均安装在主机盒内。在主机盒的主板上安装有：CPU，存储器，显示电路，汉字处理电路，国标一、二级汉字字库ROM，电源，以及键盘、盒式磁带录音机、游戏杆、扬声器、软盘驱动器、电视机、监视器等接口。此外，主板上还设有一个扩充槽口。它们在主板上的位置可以大致分为五大部分，如图2.1所示。

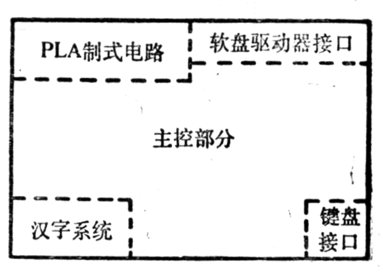


图2.1　CEC-Ⅰ主机板上各部分位置示意图

在这五大部分中，主控部分、键盘接口和PAL制式电路是CEC-Ⅰ机的最基本系统。软盘驱动器接口和汉字处理系统可看作外围扩充部分，是为了增强学习机功能而增加的功能模块。如果CEC-Ⅰ学习机的主机配上家用电视机及盒式录音机，就构成了一个基本的微机系统；若能配上监视器和软盘

驱动器，就可组成能运行DOS操作系统，与APPLLEⅡe兼容，且具有汉字支持的微机系统。

CEC-Ⅰ学习机主机的系统组成如图2.2所示。

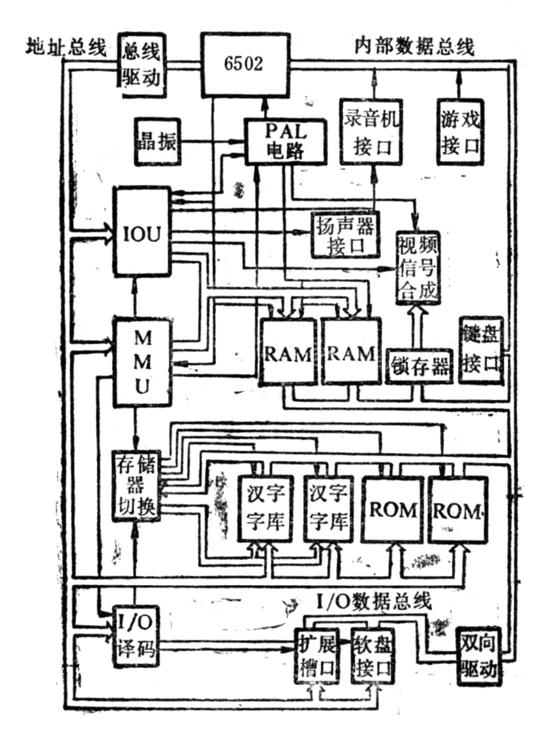


图2.2 CEC-Ⅰ 系统框图

1. 主控部分

① MPU——CEC-Ⅰ机的中央处理单元，是一个八位的6502微处理器，有八位数据线；十六位地址线，可寻址64K空间；时钟频率1.023MHz，每秒可执行50万次八位数

运算，处理能力较强。

② MMU——存储器管理器。主要控制存储器的寻址和动态RAM的刷新，同时提供外围接口卡的控制信号。

③ IOU——输入/输出管理器。控制各种输入/输出设备的接口电路，如键盘、电视机、监视器的接口；盒式磁带录音机、扬声器、游戏杆和软盘驱动器等接口。

④ PAL——逻辑定时阵列。生成系统工作时所需的各种定时时序，如Φ_0，Φ_1，$\overline{RAS}$，$\overline{CAS}$，$\overline{LDPS}$，PM，Q3，3.58M等信号。

⑤ ROM——主机上共有6片ROM。其中，2732为字符，图形转换ROM；2716为键盘数码转换ROM；一片27256ROM有16K字节，用于存放监控程序和BASIC解释程序以及游戏程序等。另一片27256固化汉字码表和汉字管理程序；两片1兆位ROM固化汉字字库。

⑥ RAM——共64K字节，由两片50464动态RAM芯片组成，每片容量为64K×4位。两片并联组成64K×8位的存储器。

此外，主控部分还有键盘接口（采用KB3600编码器）和其它I/O接口以及一个I/O扩充槽口。

2．PAL制式形成电路

我国的彩电制式是PAL制。为了配用国产彩电，所以CEC-Ⅰ机设计了PAL制式形成电路。它主要由一块TCA650(PAL色彩调制器)，一块74LS175和一块74LS146构成。利用该电路，可以在PAL制式的彩色电视机或监视器上，显示字符及高、低分辨率的彩色图形。

3．键盘接口

采用KB3600键盘编码器和ROM组成，将按键转换成相应的ASCII码。

4. 汉字系统

采用国标一、二级汉字字库（两片1兆位ROM），包括6763个汉字、符号及外文字母，一块27256ROM存放汉字码表，汉字管理程序是采用的辅存RAM区，可提供拼音、区位等输入方式。

5. 磁盘驱动器接口

主要用于实现计算机码与磁盘可识别码之间的数据转换和传送。它包括：

① 固化引导程序ROM，

② 控制逻辑，

③ 并串数据转换电路，

④ 串行数据的调制、解调电路。

2.2 6502微处理器

6502是一种8位的微处理器，具有64K的寻址能力，时钟频率为1.023MHz，即每秒可执行50万次8位数的运算，采用单一+5V电源。

2.2.1 6502的内部结构

图2.3是6502微处理器的内部结构图，它包括控制部分，寄存器部分和算术逻辑运算单元ALU三个部分。在指令执行过程中，寄存器完成一系列的数据传递和寄存。ALU是完成算术运算和逻辑运算的基本部件，控制部分则提供指令

执行过程中所需的控制信号。

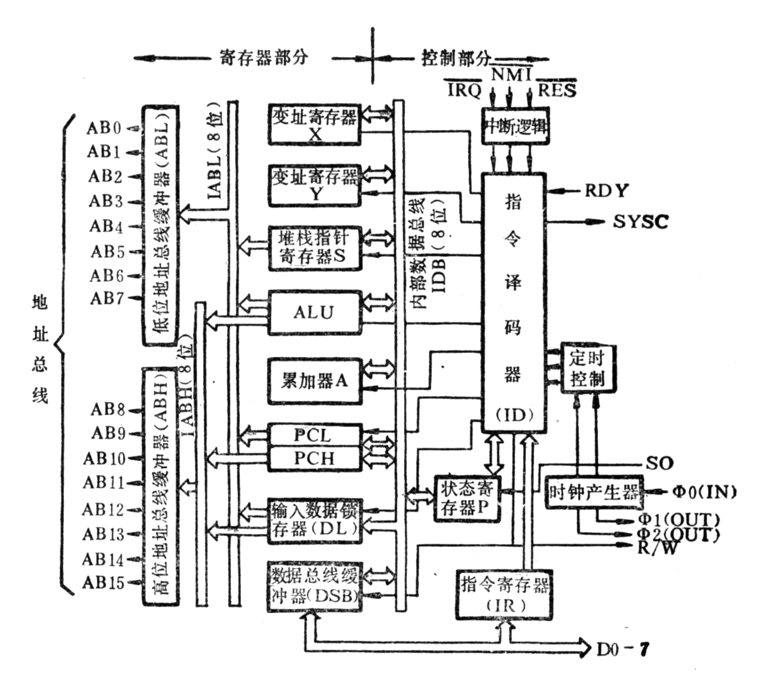


图2.3　6502的内部结构框图

1．控制部分

它是保证整个微机系统按统一时序协调操作的功能部件。它对指令进行读取、译码，然后按指令的功能发出各种控制信号，控制各部件（寄存器，存储器，ALU，I/O）执行相应的操作，以完成指令规定的功能。

2．ALU算术逻辑运算单元

它用于实现各种算术、逻辑运算，如加、减、与、或、

异或以及移位，加1，减1等操作。

3．寄存器

6502有六个可供使用的寄存器（A，X，Y，PC，S，P）。其中，除程序计数器PC是16位外，其余寄存器均为8位。

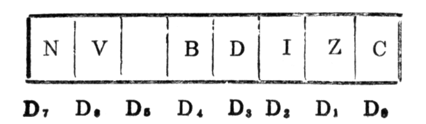
① 累加寄存器A，8位。它同ALU一起完成各种算术逻辑运算。它通常既提供ALU一个原始操作数，又存放操作结果，所以称为累加器。

② 变址寄存器X和Y，它们均为8位寄存器，主要在变址寻址方式中用来存放地址偏移量，也常被当作计数器用。此外，还可作为一般的通用寄存器，用于数据的暂存。

③ 程序计数器PC，它是6502中的唯一16位寄存器。它专司存放下一条要执行指令的地址码。当程序顺序执行时，每取出一个指令字节后PC即自动加1，为取下一个指令字节作好准备。当指令转移执行时（非顺序执行），PC中的内容将是要转移的目标地址码。

④ 堆栈指针S，是一个指示堆栈栈顶位置的8位寄存器，CEC-Ⅰ的栈顶位置为S内容加1。由于CEC-Ⅰ机是将堆栈设置在第1页存储器中，所以只需用8位寄存器指示堆栈的低8位地址。在CEC-Ⅰ的S中始终存放着紧挨栈顶的空单元的低八位地址码，所以栈顶的实际位置为S＋1。在有数据进栈时，S内容自动加1，而在数据出栈时，S内容自动减1。

⑤ 标志寄存器P，或称6502CPU的状态寄存器，8位。



N	V		B	D	I	Z	C
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

实际使用时，只用了其中7位，第5位未用。这七个标志位是：

C——进位标志。作加法运算时，若最高位有进位，则C位置1，否则置0；作减法运算时，若最高位有借位时，则C位置0，否则置1，这是与其它微机不同之处，需要特别注意。逻辑运算时，C位置0。

Z——零标志。本次操作后，若结果为0，则Z位置1，否则Z位置0。

I——中断禁止（屏蔽）标志。此位为0，表示准许中断，此位置1，表示禁止中断（非屏蔽中断不受此约束）。

D——十进制运算标志。此位置0，令ALU作二进制运算；此位置1，则令ALU作十进制运算。在6502中有专门的指令对D置0或1，详见第三章。此位状态仅对后续的加、减指令有作用。

B——BRK指令标志。执行BRK指令后此位置1，程序被中止。

V——溢出标志。若本次运算结果产生溢出，则V置1；否则V置0。

N——符号标志。它是把本次运算结果的最高位复制到N中。若V＝0（无溢出），则N代表运算结果的符号：即运算结果为负数时，N＝1；为正数时，N＝0。若V＝1（有溢出），则N代表的符号与运算结果的符号相反，即，运算结果是负数时，N＝0，正数时，N＝1。

对标志位应该掌握以下几点：

① 各条指令对标志位的影响不尽相同（详见附录）。

② 寄存器P中各标志位的状态一般是指当前指令执行后的状态，所以，标志位常常用在执行条件转移时作为条件

判断的依据。

③ 6502MPU还可用专门指令对某些标志位进行操作——置位或复位。比如CLC指令，其功能是清进位位，即对P寄存器中的C位置0，对其余各位不影响；SEC指令则是使C位置1，对其余各位不影响。

2.2.2 6502微处理器的引脚及功能

6502微处理器共有40条引脚，如图2.4所示。除电源和接地引脚外，其余38条引脚可分为三类：地址总线，数据总线和控制总线（这里不是指38条引脚本身就是总线，而是指它们分别与这三类总线相连，以下同）。

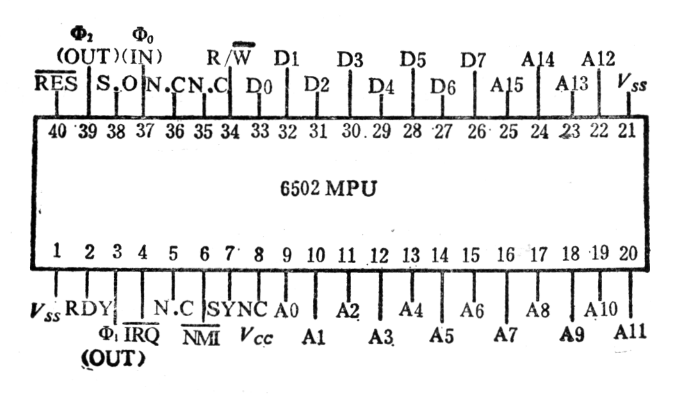


图2.4 6502引脚图

1. 地址总线A_{15}～A_0

单向输出，共16条，可寻址$2^{16}=64K(65536)$个单元。6502输出的地址在Φ_1期间内稳定。

2. 数据总线D_7～D_0

三态、双向（既可在输入方式下工作，也可在输出方式下工作），共8条，它是6502MPU与外存储器或I/O口交换数据的通道。数据总线在Φ_0期间传送数据。在Φ_1期间，数据总线缓冲器呈现高阻态，CEC-Ⅰ主机正是利用这一特性，在Φ_1期间实现显示数据的读取和动态RAM的刷新。

3．控制总线

① 时钟信号Φ_0、Φ_1、Φ_2

6502MPU有三个时钟信号Φ_0，Φ_1，Φ_2，它们为6502MPU提供工作定时。Φ_0是由晶体振荡器向MPU提供频率为1MHz左右的时钟输入信号；Φ_1和Φ_2是MPU根据Φ_0产生的两个时钟输出信号，它们是彼此反相的两个方波。6502MPU提供的这两个基本时钟信号是供存储器及外设接口使用的。在CEC-Ⅰ机中没有使用Φ_2，而是用Φ_0代替Φ_2。当$\Phi_1=1$时称作Φ_1期间；$\Phi_2=1$时，称作Φ_2期间，$\Phi_0=1$时称作Φ_0期间。

② 读/写信号$R/\overline{W}$

这是6502MPU的一个输出信号，用来控制6502对存储器或I/O口进行数据的读/写。当$R/\overline{W}$为低电平时，进行数据的写操作，即数据从MPU输出到存储器或I/O口；其余时间，$R/\overline{W}$均为高电平。$R/\overline{W}$在Φ_1期间稳定。

③ 准备就绪信号RDY

这是对6502MPU的一个输入信号。其作用是请求延长MPU的读操作周期，以适应对低速存储器或I/O口的访问。当RDY为低电平时，MPU的读操作周期可延续——即$R/\overline{W}$信号和地址线$A_{15}\sim A_0$上的电平保持不变，直到RDY变成高电平为止。但要注意，不能利用RDY延长写周期。

④ 非屏蔽中断请求（$\overline{\text{NMI}}$）

这是对MPU的输入信号。当$\overline{\text{NMI}}$有效时，即该输入信号出现从高电平到低电平的负跳变时，表明有非屏蔽中断请求，MPU执行完当条指令的操作后，就必须立即响应$\overline{\text{NMI}}$请求，转入相应的中断处理。这种中断请求是不可屏蔽的，也不受中断屏蔽标志位I的状态的影响，所以称为非屏蔽（或不可屏蔽）中断。MPU响应中断后所进行的处理过程与可屏蔽中断相似。

⑤ 可屏蔽中断请求$\overline{\text{IRQ}}$

$\overline{\text{IRQ}}$是对MPU的输入信号。当$\overline{\text{IRQ}}$为低电平时，表明外设有可屏蔽中断请求，这时如果中断屏蔽标志位I＝0，则MPU在执行完当条指令后，即转入中断响应操作；如果中断屏蔽标志位I＝1，即使$\overline{\text{IRQ}}$为低电平，MPU也不会响应外设的中断请求。在6502MPU中可以用指令SEI和CLI来设置中断屏蔽标志位I的状态，所以称$\overline{\text{IRQ}}$为可屏蔽中断请求信号。

当有可屏蔽中断请求发生时，$\overline{\text{IRQ}}$变为低电平，并维持此低电平状态直到中断请求被响应为止。

MPU对$\overline{\text{IRQ}}$和$\overline{\text{NMI}}$的采样判别是在Φ_0期间进行的。当RDY为低电平时，MPU不能响应任何中断。

⑥ 复位信号$\overline{\text{RES}}$

$\overline{\text{RES}}$是对6502的输入信号。加电时，$\overline{\text{RES}}$保持低电平，迫使MPU进入初始化状态；当＋5V电源和晶振稳定之后，$\overline{\text{RES}}$变为高电平。

⑦ 同步信号SYNC

这是6502MPU的一个输出信号。当6502处于读指令操

作码周期时，SYNC处于高电平，以识别该周期为指令的读操作码周期。

⑧ 溢出信号S.O

这是对MPU的一个输入信号。当该信号出现从高电平到低电平的负跳变时，6502的溢出标志V位被置1。

4. 其 它

V_{cc}引脚接+5V电源，电源电压允许的变动范围是±5%，极限值为+7V。

各N.C引脚均为空引脚，未用。

2.2.3 6502的时序

对存储器和I/O口的读或写是MPU最基本的操作，任何指令的执行都离不开这两种操作。所以读/写周期的时序是最基本的时序。下面我们仅就6502读/写周期的时序作一简单介绍。

1. 读周期

图2.5示出了6502MPU的标准读/写时序图。由图可以看出：在读周期内，6502在Φ_1期间输出地址和读/写信号R/$\overline{W}$（此时R/$\overline{W}$应为高电平），并在Φ_1变高后110ns内稳定，以后一直保持到Φ_0结束，并将Φ_0的下降沿作为对数据总线取数的选通信号，将数据读入MPU，要求被读存储单元输出的数据必须在Φ_0下降沿前50ns内在数据总线上稳定。

2. 写周期

如图2.5所示，在写周期内，6502MPU仍在Φ_1期间输出地址和R/$\overline{W}$信号（此时R/$\overline{W}$应为低电平）并在110ns内稳定，且保持到Φ_0结束。6502输出的数据在Φ_0变高后75ns

内稳定，并保持到Φ_0变低后30ns，在此期间被写存储单元需将数据写入。

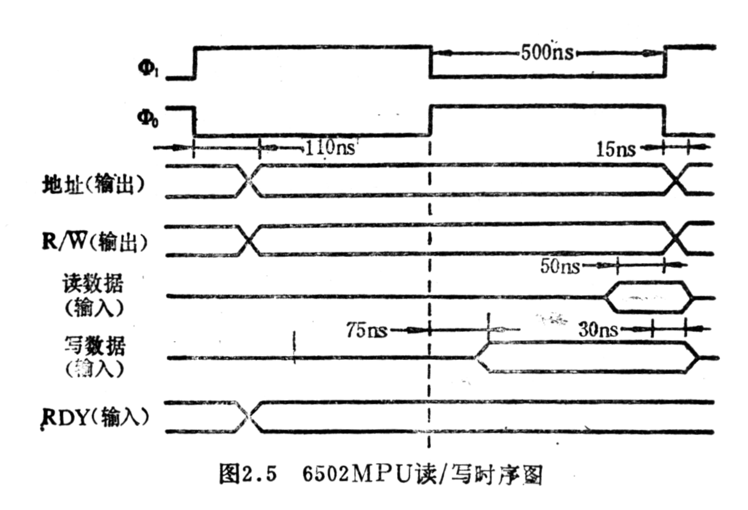


图2.5　6502MPU读/写时序图

2.3 存　储　器

6502 MPU有16条地址线，可寻址64K字节内存空间。但由于CEC-Ⅰ机采用了存储体切换技术（存储空间的映射技术），故实际可寻址96K字节，其中ROM占32K字节，RAM占64K字节。

2.3.1 CEC-Ⅰ系统的存储空间分配

CEC-Ⅰ机的内存系统分配如图2.6所示。$0000～$BFFF为主RAM的基本区域；$C000～$CFFF为输入/输出空间；$D000～$FFFF为存储体切换空间，此空间可映射到RAM，也可映射到ROM。

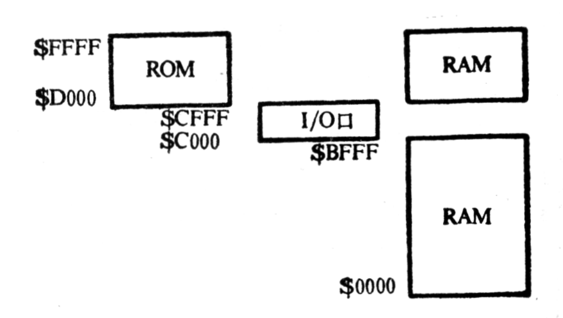


图2.6 CEC-Ⅰ内存系统分配图

2.3.2 读/写存储器

由上述可知，在CEC-Ⅰ机中有64K字节的RAM区，它是由两片HM50464存储芯片并联组成。HM50464芯片是存储容量为64K×4位的动态存储芯片，由两个芯片并联组成容量为64K×8位的存储器RAM。

1．RAM的基本性能

① 单一工作电源：+5V(±10%)。

② 低功耗：工作时350mW；备用时20mW。

③ 高速度：访问时间为150ns。

④ 数据输出由$\overline{CAS}$或$\overline{OE}$控制。

⑤ 所有信号与TTL电平兼容。

⑥ 256个刷新周期应在4ms内完成。

HM50464芯片的引脚排列如图2.7所示，各引脚功能说明如下：

$\overline{OE}$——输入，访问HM50464的使能信号。

I/O_1～I/O_4——双向，数据输入/输出引脚。

$R/\overline{W}$——输入、读/写控制信号。

$\overline{RAS}$——输入，行地址选通信号。

$\overline{CAS}$——输入，列地址选通信号。

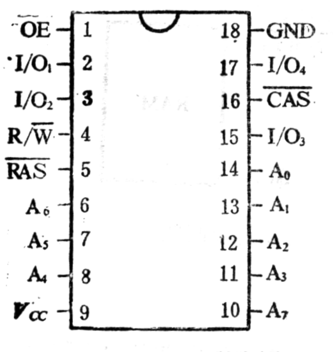


图2.7 HM50464引脚图

$A_7 \sim A_0$——行、列复用地址的输入，其中行地址又作为刷新地址输入。

V_{CC}——输入，+5V电源。

GND——电源与信号的公共地。

64K字节RAM空间被分成两大部分，其基本部分为48K字节，地址范围$0000～$BFFF；另一部分为16K字节，地址范围是$D000～$FFFF，它由存储体切换技术寻址。

RAM的绝大部分空间都提供给用户存放程序和数据，有少量页面作为系统工作区，供监控程序和BASIC解释程序使用。系统并不能阻止用户使用系统工作区，若用户要访问系统工作区时应小心谨慎，不要破坏了系统的数据，以免导致系统错误。

2．RAM的空间分配

CEC-Ⅰ机具有64K字节的RAM空间，$64K = 2^8 \times 2^8 = 256 \times 256$，故常将64K空间划分为256页（0～255），每页256个单元。其中高八位为0的页称为零页，地址范围为$0000～$00FF；高八位为1的页称为第1页，地址范围为$0100～$01FF；高八位为2的页称为第2页，地址范围为

＄0200～＄02FF，等等。图2.8所示是CEC-Ⅰ机 RAM 空间的应用分配图。

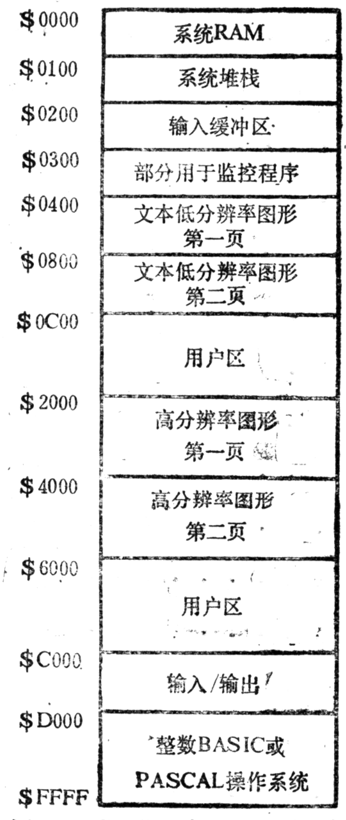


图2.8　CEC-Ⅰ机RAM的空间分配

① 零页，是系统RAM区，用于存放监控程序、BASIC解释程序和DOS文件，它们使用零页的情况如表2.1所示。在6502 MPU的指令系统中，凡零页地址都可将其地址码的

表2.1 系统程序占用零页的情况

低八位地址的高位	低八位地址的低位															
	$0	$1	$2	$3	$4	$5	$6	$7	$8	$9	$A	$B	$C	$D	$E	$F
$00	•	•	•	•	•	•					•	•	•	•	•	•
$10	•	•	•	•	•	•	•	•	•	•				•		•
$20	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$30	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$40	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$50	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$60	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$70	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$80	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$90	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$A0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$B0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$C0	•	•	•	•	•	•	•	•	•	•	•	•	•	•		
$D0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
$E0	•	•	•	•	•	•	•	•	•	•	•					
$F0	•	•	•	•	•	•	•	•	•						•	•

高八位＄00略去，只写出地址码的低八位。表中“•”表示已占用单元。

由表2.1可见，零页中只有极个别单元没有被系统程序所占用。如果你的汇编源程序要使用到零页地址时，必须格外小心，务必不要破坏了系统程序占用的单元。在迫不得已非要使用到已占用单元时，应先将要用到的零页单元的内容存放到别处，在使用完这些单元后转回到系统程序之前，再恢复这些单元（原来）的内容。

② 第1页，CEC-Ⅰ机规定第1页的256个单元（＄0100～＄01FF）作为堆栈区。所谓堆栈是指存储器中按照“先进后出”或“后进先出”原则存取数据的一个特殊区域，MPU不能随意访问这个区域的任一单元（MPU可以访问其它区域的任一单元），只能访问堆栈的栈顶单元。堆栈指针S的内容指示出当前栈顶的位置（这里只指示地址码的低八位，高八位为＄01，约定隐含），也就是指示堆栈中当前可被访问的单元。

因为CEC-Ⅰ规定第1页作为堆栈区，所以堆栈内最多只能存放256个字节的内容。当存放第257个数据时，堆栈指针就会重复，将第257个数据存放在原来第一个数据的位置上，原来第一个数据被冲掉，造成堆栈的溢出，致使程序运行出错。

③ 第2页（＄0200～＄02FF），作为键盘的一个输入行的缓冲区，共256个单元。因此，你的任何一个程序行所包含的字符个数不得超过256。

④ 第3页（＄0300～＄03FF）。CEC-Ⅰ机的监控程序和DOS使用了第3页的少数高地址单元作为向量地址，

BASIC程序中有些目标子程序使用了一部分第3页的低地址单元。监控使用第3页的详细情况见表6.2，DOS对 第3页的使用参见DOS手册。

⑤ $0400～$07FF，这1K字节不能存放用户程序与数据，它是文本显示和低分辨图形方式的第一页显示缓冲区。

$0800～$0BFF是文本显示和低分辨图形方式的第二页显示缓冲区。

⑥ $2000～$5FFF是高分辨图形显示缓冲区。当不用高分辨率图形显示时，这个区域可供用户程序使用。

无论低分辨率图形或是高分辨率图形都安排了两页显示缓冲区，这主要是为了方便用户，可以利用它们进行动画显示。

⑦ $6000～$BFFF是用户区。但在调用磁盘时，这个区域还要存放磁盘操作系统。比如，使用DOS3.3操作系统时，存储器$9600～$BFFF就被用作存放操作系统以及文件输入/输出缓冲区。

⑧ $C000～$CFFF是输入/输出设备占用的地址码区域。由于CEC-Ⅰ机的输入/输出是采用的存储器映象方式，所以它开辟了$C000～$CFFF区域作为输入/输出，所有板上的接口和要在空槽加入的接口，它们的地址都必须安排在这4K字节空间内。

⑨ $D000～$FFFF这部分是存储体切换空间。在这个地址范围内的ROM区是用于存放BASIC解释程序和监控程序；在这个地址范围内的RAM区用于存放整数BASIC或PASCAL操作系统。

3．存储体空间的切换

把不同的存储模块映射到同一地址空间称为存储体空间的切换。在CEC－Ⅰ机中，把12K地址空间＄D000～＄FFFF进行两层切换。一层是在这12K空间进行ROM和RAM之间的切换；另一层是在＄D000～＄DFFF这4K空间又进行两个4K RAM之间的切换。图2.9示出了这两层切换空间的分配图。

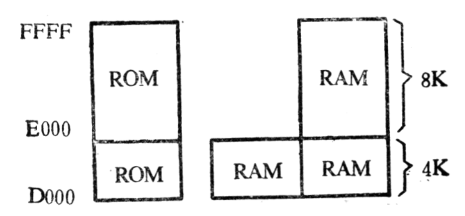


图2.9　CEC－Ⅰ机存储体切换空间分配图

2K ROM用于存放监控程序和BASIC解释程序；16K ROM（其中包含两个4K RAM的切换空间）一般用来存放整数BASIC或PASCAL操作系统的一部分。

存储体的切换是通过改变软开关的状态来实现的。通过软开关的状态可以实现三种选择：一是将＄D000～＄FFFF这12K空间切换到ROM还是RAM；二是允许写RAM还是禁止写RAM；三是将＄D000～＄DFFF这4K空间切换到第一个4K RAM还是第二个4K RAM。

表2.2列出了CEC－Ⅰ机中存储体切换的软开关地址。通过对这些地址的访问，就可设置相应的软开关，实现需要的存储体切换。

软开关地址为＄C080～＄C08F。如果软开关地址最低位为“1”，则为写RAM；为“0”时，为禁止写RAM。

表2.2　CEC-Ⅰ中存储体切换的软开关地址

软开关地址 (16进制)	写 RAM	读 RAM	读 ROM	4K RAM体 第一体	第二体
C080		•			•
C081	•		•		•
C082			•		•
C083	•	•			•
C084		•			•
C085	•		•		•
C086			•		•
C087	•	•			•
C088		•		•	
C089	•		•	•	
C08A			•	•	
C08B	•	•		•	
C08C		•		•	
C08D	•		•	•	
C08E			•	•	
C08F	•	•		•	

如果软开关地址的A_3位为“1”，则选RAM第一存储体中的4K；若为“0”，则选RAM第二存储体中的4K等等。

需要注意的是：在$D000～$DFFF空间中的两个RAM体，不能选一个读，另一个写，只能选择读、写同一个RAM体；对$D000～$FFFF空间，不能选择读其中一

部分空间的ROM，再读剩余部分的RAM，而必须照软开关设置的状态读写。对这些切换空间的访问必须仔细筹划，否则，会使程序造成极大的混乱。

CEC-Ⅰ机在加电或复位时，地址空间＄D000～＄FFFF的初始状态被设置为允许读ROM，允许写RAM，4K RAM空间在第二体。

2.3.3 只读存储器ROM

在CEC-Ⅰ主机中，采用了两块EPROM27256作为固化程序用ROM，它们在主板上的位置分别为U和U_{35}。在U_7中固化着系统程序——包括监控程序和APPLE SOFT BASIC解释程序，其程序地址为＄C100～＄FFFF。其余16K字节空间＄8000～＄BFFF和U_{35}一起用于汉字处理或其它扩充功能的固化程序。

2.4 CEC-I的输入/输出

2.4.1 输入/输出方式

1. MPU对外设的访问

在微型计算机中，除了微处理器MPU和内存储器之外，还有一个不可缺少的部分是输入/输出设备及其接口（简称I/O设备和I/O口）。最基本的输入设备是键盘，最基本的输出设备是显示器。此外还有各式打印机，盒式磁带，磁盘，CRT显示器，A/D，D/A转换器等等。由于I/O设

备种类繁多，在结构、工作速度以及数据传送方式上各不相同，所以，各I/O设备都必须通过相应的“接口”接入计算机。MPU与外设进行数据交换实际上都是通过接口来实现的。MPU通过执行指令与接口进行数据交换（外设的地址码是设在相应接口上的）；而接口与外设之间通过硬件直接传送，或用联络信号来实现数据交换，而不是通过执行指令来实现的。

一般说来，MPU对 I/O 口的访问有两种类型：

一种称为存储器对应 输入/输出 方式（或称存储器映象方式）。其特点是将外设的地址码和内存储器地址码统一编址，也就是说，$A_{15} \sim A_0$这16条地址线所包括的64K地址中，要开辟出一部分给外设用，这自然会使存储器所占容量减少。但这种方式中，MPU用于存储器的所有指令，都可用于访问 I/O 口，不再需要专门的 I/O 指令。CEC-Ⅰ学习机采用存储器映象方式，如图2.6所示，I/O 口占用的存储单元为＄C000～＄CFFF，共4K容量。

另外一种可称为专用 输入/输出 方式（或称为端口映象方式）。在这种方式中，MPU通过专门的 输入/输出 指令与 I/O 口打交道，而不能使用MPU访问存储器的各种指令。这种方式的外设不占用内存容量，如Z80 MPU就是采用的这种方式。

2. 输入/输出 方式

由于外部设备种类繁多，工作速度各异，而且对数据传送方式的要求也各不相同。因此，对外设的访问无论是采用存储器映象方式还是采用端口映象方式，在对不同外设的访问时，还有以下四种具体的 输入/输出 方式：

① 无条件传送方式

② 查询方式

③ 中断方式

④ DMA方式

DMA方式是指存储器和外设之间的直接数据传送，在数据传送过程中，不须通过MPU，所以称为直接存储器存取方式。这种输入/输出方式必须在专门的直接存取存储器控制下进行。这种方式只用于存储器与外设间需要进行大量而快速传送数据的场合（通常不用），在此讨论从略。

下面对前三种传送方式简述之。

① 无条件传送方式

无条件传送方式又称为同步传送方式。它是指MPU可以随时无条件访问外设，而不先考查外设是否已经准备好。这种传送方式最简单，硬件、软件都很节省。但它只适用于MPU与外设在数据传送时间上是同步的情况，即当MPU读外设数据时，不需考查，认定外设数据已准备好；写数据到外设时，不需考查，认定外设已空，能接收MPU送来的数据。实际上有可能还没有准备好，所以，这种数据传送方式不可靠，只能用于外设数据准备时间固定，MPU与外设同步的情况。

② 查询方式

查询方式又称为异步方式。因为外设的工作速度通常比MPU的速度慢得多，而且外设数据的准备时间通常是随机的，所以它们之间的数据交换通常是不同步的。此时就需要采用查询方式，先考查外设是否准备好。准备好了就传送；没准备好就继续考查，一直考查到准备好才传送。所以，查

询方式可以做到数据的可靠传送。但它在硬件上必须有一个状态口，以便MPU 考查外设是否准备好，软件上必须有一段查询程序。查询程序段的流程图如图2.10所示。

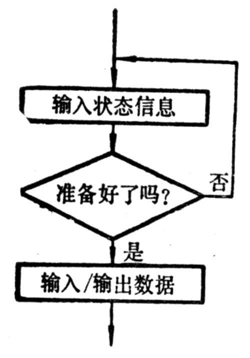


图2.10 查询程序段流程图

③ 中断方式

采用查询方式可以做到数据的可靠传送，但当外设未准备好时，MPU要不断地查询外设状态，直到外设准备好，程序才能往下执行。这样就浪费了MPU的时间，使本来是高速工作的MPU无法充分发挥出应有的效率。为了充分发挥MPU的效率，又做到数据的可靠传送，人们研究出了中断传送方式。它的基本思想是：计算机和外设启动后，MPU执行自己的程序，外设作数据传送的准备。当外设准备好时，就向MPU发出中断申请，请求与MPU交换数据。这样，在外设准备过程中，MPU没有浪费时间去考查外设是否准备好，而是在不停地执行自己的程序。如果外设准备好了，外设自动提出中断请求，请求与MPU交换数据。如果这时MPU可以响应这个请求的话（中断响应），则MPU就转入中断服务程序，实现与外设之间的数据交换。所以，中断传送方式既做到了数据的可靠传送，又不浪费MPU的时间，充分发挥MPU的效率。所以，中断是计算机中最重要的一种输入/输出方式。

2.4.2 CEC-Ⅰ的中断系统

从6502MPU的引脚图可知，6502可以有两类中断请求，一类是可屏蔽中断请求$\overline{IRQ}$，另一类是不可屏蔽的中断请求$\overline{NMI}$。这两条中断请求输入线平时都处于高电平状态，只有当有中断请求时，才会变为低电平。

当$\overline{NMI}$线上出现由高电平到低电平的负跳变边沿时，6502 MPU就会识别出非屏蔽请求。这时，6502在执行完正在执行的当前指令之后，就一定会暂停主程序的执行而进入$\overline{NMI}$中断响应周期。在$\overline{NMI}$中断响应周期中，6502将自动进行以下工作：

① 将程序计数器PC的值(即暂停的主程序断点的值)送入堆栈保存(先将PC的高字节进栈，再将PC的低字节进栈)。

② 紧接着将标志寄存器P的内容送入堆栈保存。

③ 将标志寄存器P中的中断禁止位置1，以禁止MPU再响应其它中断。

④ MPU从$03FB和$03FD单元中取出$\overline{NMI}$请求的中断服务程序的入口地址码（前者存入口地址的低八位，后者存入口地址的高八位）送PC。因此，在$\overline{NMI}$中断响应周期结束之后，MPU就转向PC内容所指示的新地址，开始运行中断服务程序，实现MPU与外设之间的数据交换。也就是说，$03FB和$03FD是存放中断服务程序入口地址的地址，常称为中断向量。中断向量是专门存放中断服务程序入口地址的地址。MPU运行完中断服务程序后，又返回主程序，继续运行主程序。

非屏蔽中断响应是MPU对$\overline{NMI}$引脚出现负跳变边沿

的响应，是不可屏蔽的，也不受P寄存器中I位状态的影响。

MPU对可屏蔽中断请求$\overline{IRQ}$的响应与$\overline{NMI}$有所不同，当MPU检测到$\overline{IRQ}$引脚出现低电平（而不是负跳变边沿），而且P寄存器中的中断禁止位I=0时，则MPU在执行完当前指令后，立即进入$\overline{IRQ}$的中断响应周期。当I=1时，则不能响应（对$\overline{NMI}$的响应不受I位状态的影响）。在$\overline{IRQ}$中断响应周期中，MPU所进行的工作与$\overline{NMI}$响应周期所做工作相似，即：

① 将程序计数器PC的值（即主程序断点的值）送入堆栈保存（先存PC的高字节，再存PC低字节）。

② 将标志寄存器P的内容入栈保存。

③ 将标志寄存器P的中断禁止位I位置1，以禁止MPU再响应其它外设提出的$\overline{IRQ}$请求（不能禁止对$\overline{NMI}$请求的响应）。

④ MPU从$\overline{IRQ}$的中断向量地址中，取出中断服务程序的入口地址送PC。这样，在$\overline{IRQ}$中断响应周期结束后，MPU就转向PC内容指示的入口地址去执行中断服务程序，实现与外设的数据交换。CEC-Ⅰ的$\overline{IRQ}$中断向量地址是$03FE和$03FF。这就是说，用$\overline{IRQ}$中断方式与MPU进行数据交换的外设，它的中断服务程序的入口地址必须存放在$03FE和$03FF中，在$03FE中存放入口地址的低八位，在$03FF中存放入口地址的高八位。

下面，我们介绍一段中断服务程序的一般模式：

主程序

```
IRQVL   EQU     $03FE          ；定义中断向量低八位。
IRQVH   EQU     $03FF          ；定义中断向量高八位。
```

```
        ⋮
LDA    #>INTSERV    ；取中断服务程序入口地址
                      的低八位。
STA    IRQVL        ；存入中断向量的低八位地
                      址所指示的单元中。
LDA    #<INTSERV    ；取中断服务程序入口地址
                      的高八位。
STA    IRQVH        ；存入中断向量的高八位地
                      址所指示的单元中。
        ⋮
```

中断服务程序

```
INTSERV  PHA   ；中断服务程序开始，保护现场（将A、X、Y
               各寄存器内容入栈保存）。
         TXA
         PHA
         TYA
         PHA
          ⋮
               }MPU与请求中断的外设交换数据。
          ⋮
         PLA   ；恢复现场（恢复A、X、Y原先存入堆栈的
               内容）。
         TAY
         PLA
         TAX
         PLA
         RTI   ；返回主程序。
```

采用中断方式与MPU交换数据的外部设备我们通称为中断源。当有多个中断源同时请求中断时，对有的中断请求必须立即响应，对有的中断请求可以稍缓响应。这样，就可以赋予不同的中断源以不同的优先权。当它们同时请求中断

时，对于优先权大的先响应；优先权小的后响应。对于不同中断源的优先权排队问题，可以采用硬件的办法，也可采用软件的办法来解决。

若用硬件的办法来解决中断响应问题，通常是用优先权编码器，比如74148，将各中断源的中断请求转换成相应的优先权代码，进行中断优先权排队，代码越大，优先权越大。用硬件方法进行优先权排队的优点是响应时间很短，缺点是要增加芯片，成本高。

若用软件办法来解决中断响应问题时，则需要安排一个中断源的查询程序。此查询程序的入口地址要先存入上述CEC-Ⅰ的中断向量地址中，当MPU响应中断后（中断响应周期结束后），就转向查询程序入口去执行中断查询程序。查询到请求中断的中断源时，就转到相应的中断服务程序去执行。因此，在软件排队中，优先权的大小决定于中断源在查询程序中的次序，先查询的，优先权大，MPU先响应；响应完了之后再继续查询后面的中断源，若有中断申请，再依次响应。

假设INTR为中断请求的状态寄存器（地址），状态寄存器的$D_7 \sim D_0$位分别作为八个中断源有无中断请求的状态标志。若某位为1，则表示该位对应的中断源有中断请求；若为0，则该中断源无中断请求。这样，查询程序的一般模式可写为：

```
FIND  PHA              ；保护现场。
      TXA
      PHA
      TYA
```

```
PHA
LDA  INTR        ；将中断状态寄存器内容送累加器A。
ASL  A           ；将累加器的最高位 A7 送标志寄存
                   器P的进位位C。
BCS  INTSERV7    ；若A7=1，则跳转到中断源7的服
                   务程序入口（INTSERV7）去执
                   行中断服务程序；若A7=0，则顺
                   序执行。
ASL  A           ；C←A6，
BCS  INTSERV6    ；若A6=1，转中断源6的服务程序入
                   口。
ASL  A           ；C←A5
BCS  INTSERV5    ；若A5=1，转中断源5的服务程序
                   入口。
 ⋮
RTI
```

2.4.3 输入/输出空间的分配

如前所述，CEC-Ⅰ机的输入/输出是采用存储器映象方式，并且分配了＄C000～＄CFFF这4K空间作为访问外设用。这4K输入输出空间大致可分为两类：一类是所谓内核输入输出，它包括键盘数据的输入，各类I/O接口的状态标志输入，选通输出，扬声器和盒式录音机接口的标志输出以及对软开关的访问等。它们占用了输入输出空间＄C000～＄C08F；另一类是外围扩展输入输出，占用的地址空间为＄C090～＄CFFF。

2.4.4 外围扩展输入输出

CEC-Ⅰ机在逻辑上可设置1～7号共七个外围扩展槽口，

占用地址＄C100～＄CFFF。在每个槽口上，可以插入专为CEC-Ⅰ机（或APPLE机）设计的扩展卡。为了使用方便，CEC-Ⅰ机为每个槽口总共安排了280个地址单元可供使用，此外还安排了2K字节的公用ROM区，每个槽口都可申请使用它。分配给每个槽口的280个单元中包括16个输入/输出单元和256个单元的扩展槽ROM区，以及8个单元的主RAM存储区（作为每个扩展卡的高速暂存器）。

1．扩展槽口的ROM区分配

CEC-Ⅰ上7个槽口的ROM区分配如下：

槽口号	地址	选通信号
1	＄C100～＄C1FF	$\overline{\text{I/O SELECT1}}$
2	＄C200～＄C2FF	$\overline{\text{I/O SELECT2}}$
3	＄C300～＄C3FF	$\overline{\text{I/O SELECT3}}$
4	＄C400～＄C4FF	$\overline{\text{I/O SELECT4}}$
5	＄C500～＄C5FF	$\overline{\text{I/O SELECT5}}$
6	＄C600～＄C6FF	$\overline{\text{I/O SELECT6}}$
7	＄C700～＄C7FF	$\overline{\text{I/O SELECT7}}$

注意：当槽口的第1引脚信号$\overline{\text{I/O SELECT}}$ n为低电平时，就作为槽口n所插扩展卡上ROM芯片的选通信号，并由其地址的低八位对该页的256个单元进行访问。

每个槽口的这256个单元，通常用来安排每个扩展卡上的只读存储器，以存放引导程序或管理外设的子程序。如果需要存放大的子程序或管理程序，CEC-Ⅰ机已在扩展卡上安排了一个公用ROM，其地址为＄C800～＄CFFF，共2K字节。通过一个公共ROM使能控制电路来控制它的使用，使它在每一时刻仅允许一个扩展卡使用。自然这2K字节的ROM也可以在每一个扩展卡上都安排一块，当CPU启动某

一槽口时，插于该槽口的扩展卡上的这2K字节ROM芯片才能选通；而其它板上的ROM则处于禁止状态，这样，就实现了分时共享＄C800～＄CFFF这2K字节的存储空间。

2．16个输入/输出单元的分配

每个槽口的分配情况如下：

槽口号	地址	选通信号
1	＄C090～＄C09F	$\overline{\text{DEVICE SELECT}_1}$
2	＄C0A0～＄C0AF	$\overline{\text{DEVICE SELECT}_2}$
3	＄C0B0～＄C0BF	$\overline{\text{DEVICE SELECT}_3}$
4	＄C0C0～＄C0CF	$\overline{\text{DEVICE SELECT}_4}$
5	＄C0D0～＄C0DF	$\overline{\text{DEVICE SELECT}_5}$
6	＄C0E0～＄C0EF	$\overline{\text{DEVICE SELECT}_6}$
7	＄C0F0～＄C0FF	$\overline{\text{DEVICE SELECT}_7}$

扩展槽口的第41号引脚信号$\overline{\text{DEVICE SELECTn}}$为低电平时，表明槽口n的输入输出单元被访问，并由地址的最低四位A_3～A_0来选择16个单元中的某一个，n为槽口号。

3．扩展卡的RAM空间

CPU在主RAM区巧妙地安排了56个单元，分别供七个扩展卡使用，每个卡8个单元，作为每个卡的高速暂存器使用，它们的地址分配如表2.3。

4．CEC-Ⅰ机上的外围扩展槽口

在CEC-I机主板上装有软盘驱动器接口和汉字处理电路，它们分别占用了6号和3号槽口地址。实际上，在CEC-Ⅰ机主板上只设有一个50线外围扩展槽口，它与APPLE Ⅱ 50线槽兼容，在逻辑上可以分别代替1号、2号、4号、5号、7号槽口使用。为此，在主板上设置了一个跨接插座J10（如图2.11所示），利用短接插塞来选择当前所用的槽号。

表2.3 扩展卡的RAM地址分配

基地址	槽口号						
（16进制）	1	2	3	4	5	6	7
0478	0479	047A	047B	047C	047D	047E	047F
04F8	04F9	04FA	04FB	04FC	04FD	04FE	04FF
0578	0579	057A	057B	057C	057D	057E	057F
05F8	05F9	05FA	05FB	05FC	05FD	05FE	05FF
0678	0679	067A	067B	067C	067D	067E	067F
06F8	06F9	06FA	06FB	06FC	06FD	06FE	06FF
0778	0779	077A	077B	077C	077D	077E	077F
07F8	07F9	07FA	07FB	07FC	07FD	07FE	07FF

若要选择某槽号，只需将两个短路插塞插在J10上对应的两排插针上。机器出厂时，一般选择1号槽口，短路插塞插在相对于1号槽的位置上。

50线扩展槽的引脚图见图2.12。

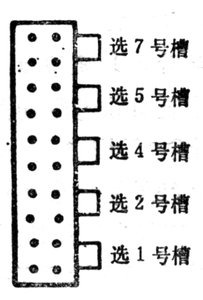


图2.11 槽号与插针位置

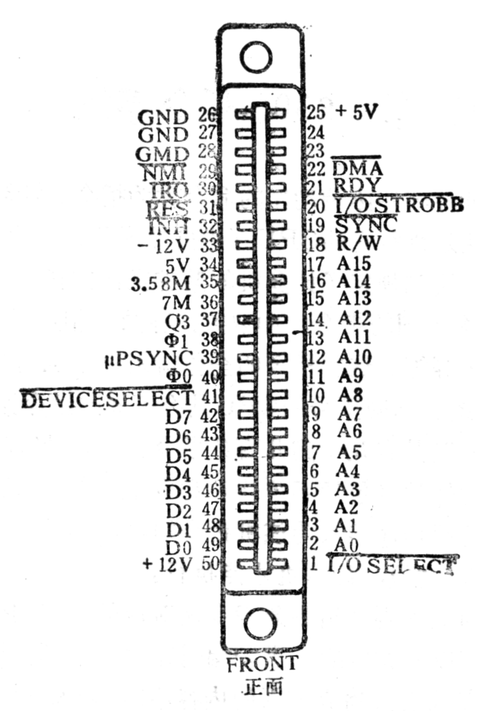


图2.12　50线外围扩展槽引脚图

第三章　6502指令系统

控制计算机各部件协调动作的命令称为指令。计算机硬件的基本功能就是执行它的CPU（中央处理单元）的指令，每一条指令都指明和规定了计算机的某种操作内容。每种型号的CPU有许多不同的指令，如取数，存数，相加，相减，移位，转移……等等。这些指令的全体就组成了这种型号CPU的指令系统。不同型号的CPU具有不同的指令系统，一种指令系统一般具有几十到几百条指令。一台计算机所能运行的任何一种软件（包括系统软件和应用软件），最后都必须通过语言处理程序转换为该机指令的机器码程序（称目标程序，参见下一章），计算机才能执行。

所谓指令的机器码是指各条指令所对应的二进制编码，任何一条指令都以其相应的二进制编码存放于机器中。如**INX**，其功能是将变址寄存器X的内容增1，(即X＋1→X)，这是一条用助记符表示的增量指令。这种用助记符表示的指令称为符号指令，它易于人们记忆、理解和掌握，但计算机不认识，机器只认识指令的二进制编码，即指令的机器码。INX指令的机器码为11101000，常书写成16进制E8（以下同）。

每条指令的内容均由两部分组成：一部分称为操作码，它指出该条指令的操作性质（即指令功能）；另一部分是操作对象，它指明参与操作的操作数本身或操作数所在的地

址，或操作数所在的地址的地址。指令如何指明操作对象，即如何取得操作数的方式，就称为指令的寻址方式。6502 CPU的各条指令的操作码均为一个字节，操作对象通常用1～2个字节表示，格式比较统一。

CEC-Ⅰ型中华学习机使用的是6502 CPU，具有简单明了的指令系统和灵活多样的寻址方式，因此，使用方便。下面，首先介绍6502 CPU的寻址方式。

3.1 6502的寻址方式

1. 立即寻址（IMMEDIATE）

在这种寻址方式中，指令的操作对象部分直接给出了操作数本身，该操作数就称为立即数。

如采用立即寻址方式的取数指令LDA # $FF。其中#表示紧跟其后的是立即数 $FF，而不是地址码，$是CEC-Ⅰ机中十六进制数的标志。该指令的功能是将立即数 $FF送累加器A，即 $FF→A，如图3.1所示。图中M表示存储器，下面是存储器中存储的指令机器码和立即数FF。

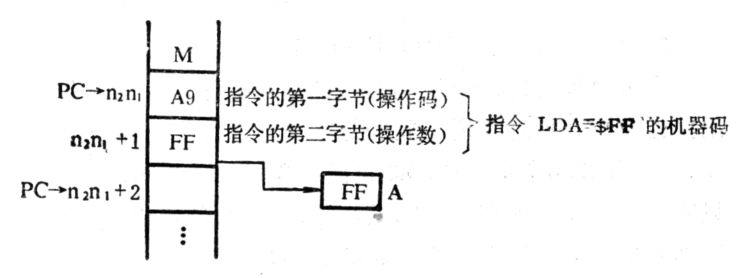


图3.1 采用立即寻址的LDA功能示意图

2．绝对寻址（ABSOLUTE）

在这种寻址方式中，指令的操作对象部分不是操作数自身，而是操作数所在地址的地址码（称为绝对地址或直接地址）。例如，采用绝对寻址方式的取数指令LDA ＄0C00，是把绝对地址为＄0C00单元的内容取出来送累加器A，即（＄0C00）→A，如图3.2所示。在这里＄0C00表示绝对地址，（＄0C00）表示＄0C00单元存储的内容（以下表示法同）。

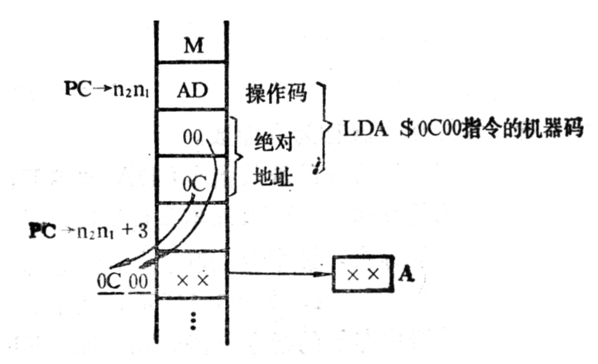


图3.2　采用绝对寻址方式的LDA功能示意图

3．零页寻址（ZERO PAGE）

零页寻址与绝对寻址类似，都是在操作对象部分直接给出操作数所在地址，所不同的是零页寻址方式的寻址范围只能在零页的256个单元中，即只能在＄0000～＄00FF中，故只需给出操作数所在地址的低八位，其高八位皆约定为＄00。比如采用零页寻址方式的取数指令LDA ＄0C。该指令在操作对象部分只给出了操作数所在地址的低八位＄0C，这就

表明操作数所在的实际地址是＄000C单元。该指令功能是把＄000C单元中的内容取出来送累加器A，即（＄0C)→A，如图3.3所示。

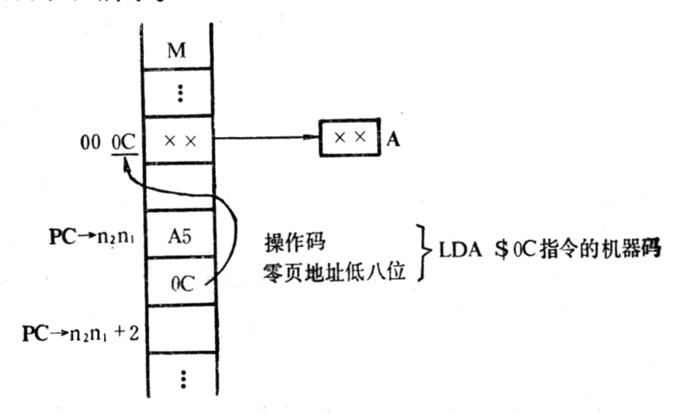


图3.3 采用零页寻址的LDA功能示意图

4．累加器寻址（ACCUM）

这种寻址方式的操作数在累加器中。如循环左移指令ROL A，其功能为：把累加器A中的操作数连同进位位C循环左移一位，如图3.4所示。

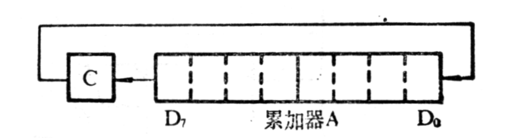


图3.4 ROL A指令功能示意图

5．隐含寻址（IMPLIED）

在这种寻址方式中，操作数所在地址隐含于操作码之中，故采用隐含寻址的指令均为一字节指令。它与累加器寻

址相似，区别仅在于：累加器寻址的操作数是在累加器A中；隐含寻址的操作数是在寄存器X或Y，S，P中。如指令DEY，其机器码为CA，一个字节；其功能为：将Y寄存器中的内容取出减1后，再送回Y寄存器，即Y－1→Y。

6．绝对变址寻址

它包括使用X寄存器的绝对变址和使用Y寄存器的绝对变址。

① 绝对X变址（ABS,X），它是使用X变址寄存器进行变址的一种寻址方式。在这种寻址方式中，是把指令中给出的16位地址作为基地址与偏移量（即X变址寄存器中的内容）相加后所得的地址，再把这个地址作为操作数实际存放的有效地址。

如采用绝对X变址的取数指令LDA ＄0340，X,（设X寄存器中的内容为＄0B），其功能是把＄0340＋X，即＄034B单元中的内容取出来送累加器A，如图3.5所示。又

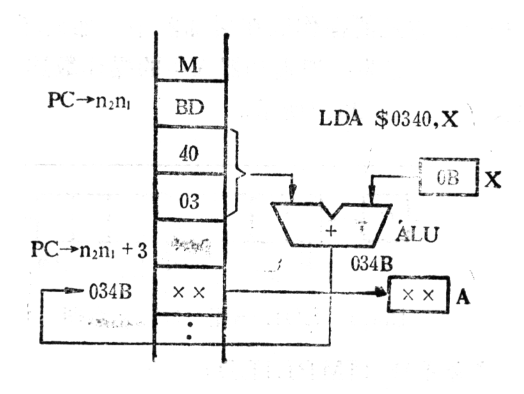


图3.5 采用绝对X变址的LDA功能示意图

如存数指令STA $0340，X，其功能是把累加器A中内容取出来存放到存储器$0340+X单元中去。

② 绝对Y变址（ABS，Y），其寻址方式与绝对X变址相似，其区别仅在于绝对Y变址中使用的变址寄存器是Y寄存器。如LDA $6030，Y，其功能是把$6030+Y单元中的内容取出来送累加器A。

7．零页变址寻址

它包括零页X变址（Z·PAGE，X）和零页Y变址（Z·PAGE，Y）。它们与绝对X(Y）变址类似，其区别仅在于零页变址寻址中只给出了一个8位地址（零页地址的低八位），其高八位约定为$00。将零页地址作为基地址与X(Y）寄存器的内容相加即得操作数存放的实际地址。如指令LDA $04，X，其功能为把存储器$0004+X单元中的内容取出，送累加器A；指令LDA $0A，Y，是把存储器$000A+Y单元中的数取出送累加器A。

8．间接寻址（INDIRECT）

在这种寻址方式中，指令的操作对象部分给出的是操作数所在地址的地址(称为间接地址)。如无条件转移指令JMP（$100A），其功能首先从指令所指出的间接地址$100A单元中取出操作数有效地址的低八位（假设为$A5），从间接地址加1单元（即$100A+1单元）中取出操作数有效地址的高八位（假设为$20），则$20A5即为操作数的实际地址；然后将$20A5送程序计数器PC，CPU随即就转向PC内容所指示的单元（$20A5单元）去执行指令。上述过程如图3.6所示。

9．先变址（X）间接寻址（IND，X）

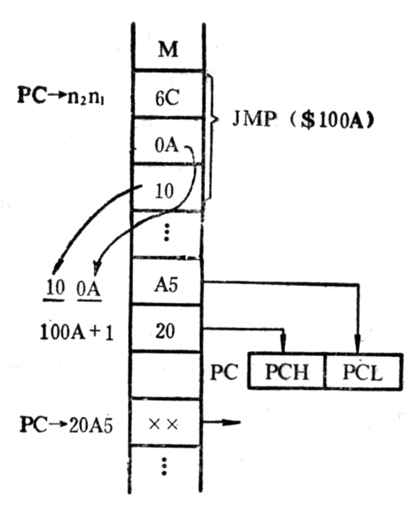


图3.6 JMP（$100A）间接寻址方式示意图

它是零页X变址和间接寻址两种方式的结合。首先，它以X作为变址寄存器，将零页中的基地址IND和X中的内容相加得IND+X；再以IND+X作为间接地址，从IND+X单元和IND+X+1单元中取出操作数的有效地址。如指令LDA（$0A，X），其执行过程为：首先由$0A+X(设X中内容为$02）和$0A+X+1得到零页中的地址码$0C和$0D，然后再分别把$0C单元的内容(设为$EE）和$0D单元的内容(设为$0F)取出来组成一个16位地址$0FEE，此即操作数实际存放的有效地址；最后将有效地址$0FEE中的内容取出送累加器A。上述过程如图3.7所示。

需要注意的是：如果没有特别说明，多字节数的高位字节总是存放在高位地址，反之亦然。比如图3.7中的16位地址码$0FEE，其高八位0F存放在$000D单元，低八位EE

存放在＄000C单元。在以后的叙述中仍然如此，不再说明。

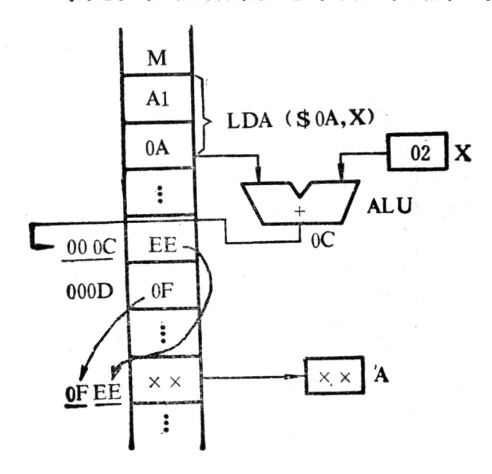


图3.7　先变址（X）间接寻址示意图

10．后变址（Y）间接寻址((IND)，Y)

这是间接寻址和绝对Y变址方法相结合的一种寻址方式。其寻址方法是先在零页中作间接寻址，取到一个16位基地址，然后再进行绝对Y变址寻址，即以Y寄存器为变址寄存器，将基地址与Y的内容相加得出操作数实际存放的有效地址。所以，这种寻址方式准确地说，应称为先间接寻址后绝对Y变址。

如指令LDA(＄08),Y,设Y中内容为＄02,零页中＄08单元的内容为＄35，＄09单元中的内容为＄0F。则该指令的功能是：首先从指令指定的零页地址＄08和＄08＋1单元中分别取出它们的内容合成为一个16位地址＄0F35，然后将此地址作为基地址与Y中内容相加得：＄0F35＋＄02＝＄0F37，此即操作数实际存放的有效地址，最后从有效地

址＄0F37单元中取出操作数送累加器A。上述过程如图3.8所示。

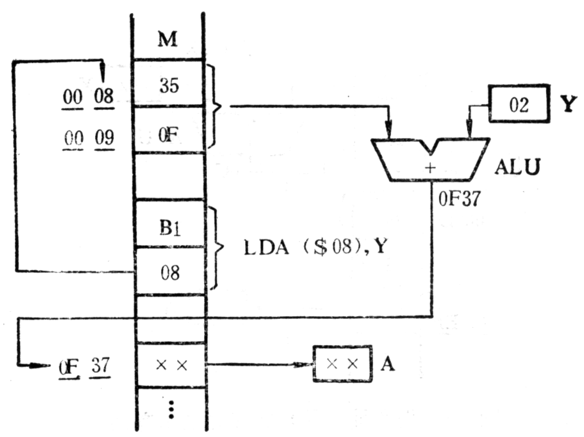


图3.8 LDA（＄08），Y的寻址方式示意图

11. 相对寻址（RELATIVE）

这种寻址方式用于相对转移指令中，指令长度为二字节，如图3.9(*a*)所示。图中第二字节是相对偏移量D，用带

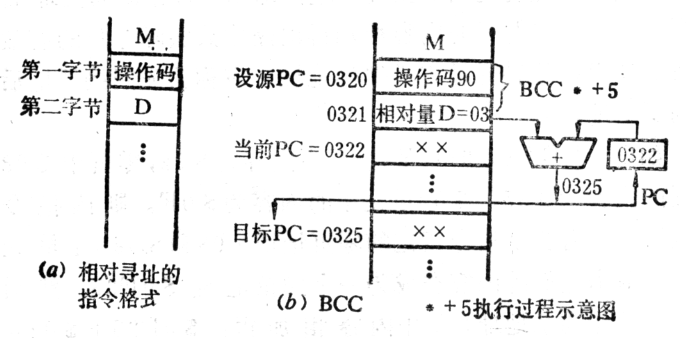


图3.9 相对寻址方式

符号数的补码表示，故其所能表示的偏移量范围为－128～＋127。

在相对寻址方式中，是将本条指令操作码所在字节的地址（常称源首址，在图3.9(*b*)中用＊表示）与绝对偏移量d（又称跳转步长）相加即得操作数的有效地址（常称为目标地址）。绝对偏移量d＝目标地址－源首址，而相对偏移量D＝d－2，故相对转移中允许的跳转步长d＝D＋2＝－126～＋129，即从源首址往低字节地址方向最大只能跳转126个字节；往高字节地址方向最大只能跳转129个字节。

这里需要指出的是：指令机器码的第二字节不是用绝对偏移量d，而是用相对偏移量D。这是因为采用相对寻址方式的指令均为二字节指令，故CPU在取完指令码的两个字节后，当前PC的内容已等于源PC内容（源首址）＋2，故目标地址＝源PC＋d＝源PC＋2＋D＝当前PC＋D。如指令BCC＊＋5，其指令功能为：当标志寄存器中的进位位C＝0时，CPU就转移到＊＋5单元去执行指令；否则顺序执行指令。指令中的＊号表示本条指令第一字节所在地址，即源地址。

以上十一种（细分为十三种）寻址方式并不是每条指令都具有。每条指令具有哪些寻址方式，请查阅6502指令表。编程时可根据程序的具体要求灵活选用某条指令所允许的某一种寻址方式，以达到程序占用内存少，执行速度快的目的。

一般说来，累加器寻址、隐含寻址方式中，操作数是在CPU的内部寄存器中，因而取操作数不必访问存储器，所以执行速度快。零页寻址方式占用内存少，执行速度快，所以

监控程序、BASIC解释程序以及DOS都大量地使用零页，以提高程序的效率。

变址寻址方式常用于处理数据块。间接寻址方式常用来存放专用程序的入口地址。相对寻址是用偏移量（跳转步长）来指出跳转的目标地址，这就减少了指令的字节长度，缩短了指令的执行时间。立即数寻址常用于设置初始数据。

需要注意的是：在遇到变址计算的寻址方式中，有些寻址方式是不许跨页的，如零页X(Y)变址寻址和先变址(X)间接寻址，它们都只能在零页中进行变址计算；而绝对X(Y)变址，后变址（Y）间接寻址，则允许跨页。

3.2 6502指令及其功能

CPU6502共有56条指令，13种寻址方式。指令格式整齐，第一字节均为操作码，其后是操作数或操作数所在地址，或操作数所在地址的地址。由于每条指令可以对应不同的几种寻址方式，因此一条指令实际上相当于几条指令。这样，6502实际上应该说共有151条指令，详见6502指令系统表。

对指令的了解应包括以下几方面：

第一，指令的功能；

第二，执行该指令后对标志寄存器P中各位的影响；

第三，指令的字节数和执行指令所需的时钟周期。

下面我们将对6502指令系统进行归纳分类，并逐一介绍其功能和对标志寄存器P中各标志位的影响。各条指令的字节数和执行指令所需的时钟周期数，请参见6502指令系

统表。

3.2.1 传送指令

1. LDA——往累加器A传送的指令

这种指令的功能是将一个立即数或将某个存储单元的内容送累加器A，常表示为M→A。它包括如表3.1的八种寻址方式。

LDA指令执行后，对标志寄存器P中的Z和N两个标志位产生影响。如果传送到A中的数，其最高位是1，则置P寄存器的N位为1，最高位是零，则置N位为0，所以常称N位为符号位；如果传送到A中的数是各位全零，则置Z位为1；否则置Z位为0，所以常称Z位为零标志位。

2. LDX——往寄存器X传送的指令

该指令是将一个立即数或某个存储单元的内容送寄存器X，其功能常表示为M→X。它包括如表3.2的五种寻址方式。

3. LDY——往寄存器Y传送的指令

与往寄存器X传送的指令类似，其功能可表示为M→Y，亦包括五种寻址方式，见表3.3。

上述LDX指令和LDY指令执行后都只对P寄存器中的N位和Z位产生影响。

4. STA——将累加器A中的内容送存储器

这种形式常表示为A→M。它有七种寻址方式，见表3.4。

执行STA指令后对标志寄存器P中各位均无影响，即P中各位保持STA指令执行前的状态。

5. STX——将变址寄存器X的内容送存储器

表3.1 M→A的寻址方式

指令格式	功能	寻址方式	备注
LDA #$n	$n→A	立即数寻址	$n是八位二进制数的十六进制表示，#$n是立即数。
LDA $n	($n)→A	零页寻址	$n是零页的低八位地址
LDA $$n_2n_1$	($$n_2n_1$)→A	绝对寻址	$$n_2n_1$是操作数的绝对地址
LDA $$n_2n_1$, X	($$n_2n_1$+X)→A	绝对X变址	$$n_2n_1$是基准地址，X中内容是变址寻址方式中的偏移量
LDA $$n_2n_1$, Y	($$n_2n_1$+Y)→A	绝对Y变址	$$n_2n_1$是基准地址，Y中内容是变址寻址方式中的偏移量
LDA $n, X	($n+X)→A	零页X变址	$n是零页基准地址，X中内容是零页变址方式中的偏移量
LDA ($n, X)	(有效地址)→A	先变址(X)间接寻址	先对零页进行X变址，得到$n+X，再从零页$n+X和$n+X+1单元中取出有效地址
LDA ($n), Y	(有效地址)→A	后变址(Y)间接寻址	首先从零页$n和$n+1单元取出内容作基地址，然后将此基地址+Y，即得有效地址

表3.2　M→X的寻址方式

指令格式	功　能	寻址方式	备　　注
LDX　#$n	$n→X	立即数寻址	#$n表示立即数
LDX　$n	($n)→X	零页寻址	$n表示零页地址
LDX　$$n_2n_1$	($$n_2n_1$)→X	绝对寻址	$$n_2n_1$为绝对地址
LDX　$n,Y	($n+Y)→X	零页Y变址	$n为零页基地址，Y中内容为偏移量
LDX　$$n_2n_1$,Y	($$n_2n_1$+Y)→X	绝对Y变址	$$n_2n_1$为基地址，Y中内容为偏移量

表3.3　M→Y的寻址方式

指令格式	功　能	寻址方式
LDY　#$n	$n→Y	立即数寻址
LDY　$n	($n)→Y	零页寻址
LDY　$$n_2n_1$	($$n_2n_1$)→Y	绝对寻址
LDY　$n,X	($n+X)→Y	零页X变址
LDY　$$n_2n_1$,X	($$n_2n_1$+X)→Y	绝对X变址

表3.4　A→M的寻址方式

指令格式	功　能	寻址方式	备　注
STA　$\$n$	A→($\$n$)	零页寻址	将A中内容送零页$\$n$单元中
STA　$\$n_2n_1$	A→($\$n_2n_1$)	绝对寻址	将A中内容送绝对地址$\$n_2n_1$单元中
STA　$\$n,X$	A→($\$n+X$)	零页X变址	将A中内容送零页$\$n+X$单元中
STA　$\$n_2n_1,X$	A→($\$n_2n_1+X$)	绝对X变址	将A中内容送$\$n_2n_1+X$单元中
STA　$\$n_2n_1,Y$	A→($\$n_2n_1+Y$)	绝对Y变址	将A中内容送$\$n_2n_1+Y$单元中
STA　($\$n,X$)	A→(有效地址)	先变址(X)间接寻址	从零页$\$n+X$和$\$n+X+1$单元中取出有效地址，并将A中内容送往有效地址
STA　($\$n$), Y	A→(有效地址)	后变址(Y)间接寻址	先从零页$\$n$和$\$n+1$单元中取出内容作基准地址，将此地址与Y中内容相加，即得A中内容要送往的有效地址

这种形式常表示为X→M。它有三种寻址方式，见表3.5。

表3.5 X→M的寻址方式

指令格式	功能	寻址方式
STX $n	X→($n)	零页寻址方式
STX n_2n_1	X→(n_2n_1)	绝对寻址方式
STX $n,Y,	X→($n+Y)	零页Y变址寻址方式

6．STY——将变址寄存器Y的内容送存储器

这种形式常表示为Y→M。它也有三种寻址方式：零页寻址，绝对寻址和零页X变址寻址，情况与STX类似，不再赘述。

上述STX指令和STY指令均不影响标志寄存器P的状态。

下面是寄存器和寄存器之间的传送，这类指令皆为隐含寻址方式，用于6502CPU内部的寄存器之间的信息交换。

1．TAX——将累加器A中内容送入变址寄存器X，即A→X。

2．TXA——将变址寄存器X的内容送累加器A，即X→A。

3．TAY——将累加器A的内容送入变址寄存器Y，即A→Y。

4．TYA——将变址寄存器Y的内容送入累加器A，即Y→A。

5．TSX——将堆栈指针S的内容送入变址寄存器X，即S→X。

6．TXS——将变址寄存器X的内容送入堆栈指针S，即X→S。

前五条指令执行后均要改变标志寄存器P的Z位和N位标志，但不改变P的其余各位标志；第六条指令不影响P的任何标志位。

由上述各传送指令可见，立即数和任一存储单元的内容都可以通过不同的寻址方式，直接往寄存器A，X，Y传送；寄存器A，X，Y的内容也可以直接往存储器传送；寄存器A和X之间，A和Y之间，X和S之间也可以互相直接传送；但存储器各单元之间却不能直接交换数据，这是必须注意的。

例如，存储器＄3000单元和＄4000单元之间要交换数据，只能通过寄存器A，X，Y转送。可用以下指令来实现：

```
LDA    $3000;    ($3000)→A
STA    $4000;    A→($4000)
```

3.2.2 置标志位指令

CLC——清除进位标志C，即0→C。

SEC——置位进位标志C，即1→C。

CLD——清除十进制运算标志D，即0→D。

SED——置位十进制运算标志D，即1→D。

CLV——清除溢出标志V，即0→V。

CLI——清除中断禁止标志I，即0→I。

SEI——置位中断禁止标志I，即1→I。

以上指令均为隐含寻址方式。

3.2.3 算术运算指令

1．ADC——带进位的加法指令

即A＋M＋C→A。其中M代表某一存储单元的内容或某一立即数，C为执行本指令之前标志寄存器中的进位位的状态。ADC指令有八种寻址方式，见表3.6。

2．SBC——带进位的减法指令

指令功能可表示为$A - M - \overline{C} \rightarrow A$。M、C的含义同ADC指令。它也有与ADC指令相同的八种寻址方式，见表3.7。

表3.6 A＋M＋C→A的寻址方式

指令格式	功　　能	寻址方式	备　　注
ADC #$\$n$	A＋$\$n$＋C→A	立即数寻址	#$\$n$为立即数
ADC $\$n$	A＋($\$n$)＋C→A	零页寻址	$\$n$为零页地址
ADC $\$n_2n_1$	A＋($\$n_2n_1$)＋C→A	绝对寻址	$\$n_2n_1$为绝对地址
ADC $\$n$, X	A＋($\$n$＋X)＋C→A	零页X变址	$\$n$＋X为零页地址
ADC $\$n_2n_1$, X	A＋($\$n_2n_1$＋X)＋C→A	绝对X变址	
ADC $\$n_2n_1$, Y	A＋($\$n_2n_1$＋Y)＋C→A	绝对Y变址	
ADC ($\$n$, X)	A＋(有效地址)＋C→A	先变址(X)间接寻址	从零页$\$n$＋X和$\$n$＋X＋1单元中取出有效地址
ADC ($\$n$), Y	A＋(有效地址)＋C→A	后变址(Y)间接寻址	先从零页$\$n$和$\$n$＋1单元中取出基地址，再由基地址＋Y得出有效地址

表3.7 A-M-C 的寻址方式

指令格式	指令功能	寻址方式
SBC #$n	$A-\$n-\overline{C}\rightarrow A$	立即寻址方式
SBC $n	$A-(\$n)-\overline{C}\rightarrow A$	零页寻址方式
SBC $$n_2n_1$	$A-(\$n_2n_1)-\overline{C}\rightarrow A$	绝对寻址方式
SBC $n,X	$A-(\$n+X)-\overline{C}\rightarrow A$	零页X变址寻址
SBC $$n_2n_1$,X	$A-(\$n_2n_1+X)-\overline{C}\rightarrow A$	绝对X变址寻址
SBC $$n_2n_1$,Y	$A-(\$n_2n_1+Y)-\overline{C}\rightarrow A$	绝对Y变址寻址
SBC ($n,X)	A-(有效地址)-$\overline{C}\rightarrow$A	先变址(X)间接寻址
SBC ($n),Y	A-(有效地址)-$\overline{C}\rightarrow$A	后变址(Y)间接寻址

以上ADC和SBC指令执行后，标志寄存器P中除I，D两个状态标志不受影响外，其余状态标志都要根据相加或相减的结果而变动。

6502 CPU只有上述带进位的加、减运算指令，它们特别适合多字节数的加、减运算。6502没有不带进位的加、减运算，因此，如果要进行单字节数的加、减运算时，可在ADC指令之前预置进位标志C=0，在SBC指令之前预置$\overline{C}=0$(因为在6502CPU中，是用$\overline{C}$表示借位，这与其它CPU大都直接用C位状态表示借位不同，请注意这个特点)。

例1. 求两个单字节二进制数$3B和$2D之和，并将

结果存入＄6000单元。

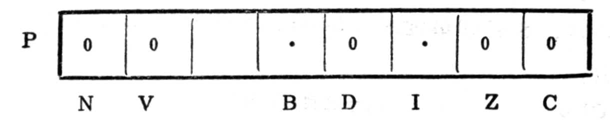
解：因为这里是要求两个单字节二进制数之和，故在加法指令ADC之前应先预置进位标志C＝0。源程序如下：

```
LDA  #$3B   ;   $3B→A
CLD         ;   预置十进制运算标志D=0，表明下面的
                运算是二进制运算，而不是十进制运算
CLC         ;   预置进位标志C=0
ADC  #$2D   ;   A+$2D+C=$3B+$2D+0→A
STA  $6000  ;   A→($6000)，结果存$6000单元
BRK         ;   结束
```

对标志位的影响可作如下分析：

$$
\begin{array}{rr}
 & \$3B = 00111011 \\
 & \$2D = 00101101 \\
+ & \boxed{C} = \quad\quad\quad 0 \\
\hline
 & \$68 = 01101000
\end{array}
$$

由于相加结果最高位无进位，故进位标志C＝0；由于结果非零，故零标志位Z＝0；由于结果最高位为0，故符号位N＝0；由于结果＄68没有超出一个字节所能表示的最大正数，故溢出位V＝0。对B位和I位都没有影响，此结果可表示为：



例2．已知A中内容为02，（＄06）＝03，求A－（＄06）＝？（十进制运算）

解：这里也是两个单字节数的运算，故需要先预置

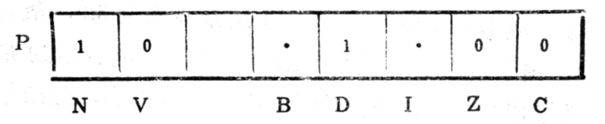
$\bar{C}=0$，(即预置C＝1)，源程序如下：

```
SED          ;   预置D=1，表示下面进行的是十进制数运算
SEC          ;   预置C=1，则C̄=0
SBC  $06     ;   A-($06)-C̄→A
BRK          ;   结束
```

上述直接运算为：

$$
\begin{array}{rl}
02 = & 00000010 \\
03 = & 00000011 \\
- \quad \bar{C} = & \qquad\quad 0 \\
\hline
\boxed{1} & 11111111 = [-1]_{补}
\end{array}
$$

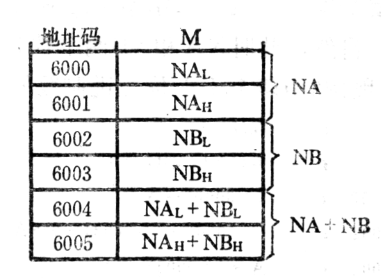
可见，最后结果是－1，它在计算机中以补码形式出现，即$[-1]_{补}=11111111$。小方框中的内容为借位$\bar{C}=1$。运算结果对标志位的影响是：因为结果的最高位是1，故N＝1(表明结果是负数)；因为结果非0，故Z＝0；因为有借位（$\bar{C}=1$），故进位位C＝0；因为无溢出，故V＝0，此外，对B位和I位没有影响。上述对标志寄存器各位的影响可表示为：



例3．设两个16位二进制数NA和NB分别存放在$6001、$6000单元和$6003、$6002单元中，如图3.10所示。

```
CLD          ;   清十进制运算标志
CLC          ;   清进位标志
LDA  $6000   ;   ($6000)→A
ADC  $6002   ;   ($6000)+($6002)+0→A
STA  $6004   ;   A→($6004)
```

```
LDA   $6001  ;    ($6001)→A
ADC   $6003  ;    ($6001)+($6003)+C→A
STA   $6005  ;    A→($6005)
BRK          ;    结束
```



地址码	M	
6000	NA_L	NA
6001	NA_H	
6002	NB_L	NB
6003	NB_H	
6004	NA_L+NB_L	NA+NB
6005	NA_H+NB_H	

图3.10　实现NA+NB的源程序

该程序实现了将两个分别存于＄6001，＄6000单元和＄6003，＄6002单元中的十六位二进制数NA和NB相加，并将结果存入＄6005，＄6004单元中（高位字节存高位地址，低位字节存低位地址）。由程序可以看出，在执行两个数的低八位相加（NA_L+NB_L），即执行指令ADC ＄6002之前，是用CLC指令将进位C清0的，以免C中原来的值影响本次运算的正确结果；但在进行两个数的高八位相加NA_H+NB_H，即执行指令ADC ＄6003之前，却不能再清进位，因为此时C中内容是低八位相加后的结果，它的进位应参加高八位的相加运算。因为6502 CPU只有带进位的加法指令ADC和带借位的减法指令SBC，没有不带进位（和借位）的加减指令。因此，在进行加、减运算之前，要根据具体要求对进（借）位进行正确处理，以求得到正确的结果。

此例中是进行的二进制数相加，故进行加法运算之前用CLD指令将十进制运算标志D置0；如果是进行十进制数相

加、减的运算，则在进行加（减）运算之前，要用SED指令将十进制运算标志D置1，以表明下面进行的运算是十进制运算。所以，如果将上述程序中的CLD指令换成SED指令，则上述程序就是实现将两个双字节十进制数（四位BCD码）相加的运算，自然，此时NA和NB两个数都应是十进制数。

例4．实现例3中两个十六位二进制数的减法。

```
CLD          ;    置D=0
SEC          ;    置C=1，则C̄ =0
LDA   $6000  ;    ($6000)→A
SBC   $6002  ;    ($6000)-($6002)-0→A
STA   $6004  ;    A→($6004)
LDA   $6001  ;    ($6001)→A
SBC   $6003  ;    ($6001)-($6003)- C̄→A
STA   $6005  ;    A→($6005) 存结果
BRK          ;    结束
```

该程序实现了两个十六位二进制数NA－NB的运算。在进行低八位相减之前，应预置$\overline{C}=0$，即预置C＝1，故这里使用了SEC指令，置C＝1。在进行高八位相减运算前，则不应预置C的状态。这就是用ADC指令和SBC指令进行多字节数的加、减运算时的方便之处。

例5．求两个十进制数相减。

```
SEC          ;    预置C=1，即 C̄ =0
SED          ;    预置D=1，表明下面是进行十进制运算
LDA   $6000  ;    ($6000)→A
SBC   $6001  ;    ($6000)-($6001)-C̄→A
STA   $6002  ;    A→($6002)，存结果
BRK          ;    结束
```

在＄6000和＄6001单元中存放的应是十进制数。

3. INC——增量指令

这条指令使存储单元内容增1，即M＋1→M。INC指令有四种寻址方式，执行结果影响N，Z两个标志位，见表3.8。

表3.8 INC指令寻址方式

指令格式	功能	寻址方式
INC $n	($n)+1→($n)	零页寻址方式
INC $n_2n_1	($n_2n_1)+1→($n_2n_1)	绝对寻址方式
INC $n, X	($n+X)+1→($n+X)	零页X变址寻址
INC $n_2n_1, X	($n_2n_1+X)+1→($n_2n_1+X)	绝对X变址寻址

4. DEC——减量指令

本指令使存储单元内容减1，即M－1→M。它具有与INC指令相同的四种寻址方式，执行结果影响N，Z两个标志位，见表3.9。

表3.9 DEC指令寻址方式

指令格式	功能	寻址方式
DEC $n	($n)－1→($n)	零页寻址方式
DEC $n_2n_1	($n_2n_1)－1→($n_2n_1)	绝对寻址
DEC $n, X	($n+X)－1→($n+X)	零页X变址寻址
DEC $n_2n_1, X	($n_2n_1+X)－1→($n_2n_1+X)	绝对X变址寻址

下述四条指令皆为隐含寻址方式的单字节指令，影响N，Z两个标志位。

INX——变址寄存器X的内容增1，即X+1→X。

DEX——变址寄存器X的内容减1，即X-1→X。

INY——变址寄存器Y的内容增1，即Y+1→Y。

DEY——变址寄存器Y的内容减1，即Y-1→Y。

3.2.4 比较指令

1. CMP ——累加器与存储器比较

CMP指令作不带进位的减法操作，即A-M，运算结果不送A，只影响P中的标志位C，N，Z。利用执行CMP指令后P中的C，N，Z标志位的状态就可以判断A和M中的内容是否相等或谁大谁小。比如，若Z=1，则表示A=M；若C=1，则表示够减无借位，即A≥M；若C=0，则表示不够减，有借位，即A＜M。CMP指令如同减法指令一样，具有与SBC指令相同的八种寻址方式。

2. CPX——X变址寄存器与存储器比较

3. CPY——Y变址寄存器与存储器比较

CPX或CPY指令的功能与CMP相似，不同的只是把累加器A换成了变址寄存器X或Y；此外，CPX和CPY只有三种寻址方式，以CPX为例，其格式见表3.10。

比较指令常与条件转移指令配合使用，实现程序的条件转移。

3.2.5 逻辑运算指令

1. AND——逻辑“与”指令

表3.10 CPX指令寻址方式

指令格式	功能	寻址方式
CPX #$n	X－$n	立即数寻址
CPX $n	X－($n)	零页寻址
CPX $$n_2n_1$	X－($$n_2n_1$)	绝对寻址

AND指令将累加器A的内容与M的内容相“与”后，结果送A，即A∧M→M。M代表某一立即数或某一存储单元的内容。

AND指令具有八种寻址方式，其格式见表3.11。

表3.11 AND指令寻址方式

指令格式	功能	寻址方式
AND #$n	A∧$n→A	立即寻址
AND $n	A∧($n)→A	零页寻址
AND $$n_2n_1$	A∧($$n_2n_1$)→A	绝对寻址
AND $n，X	A∧($n＋X)→A	零页X变址
AND $$n_2n_1$,X	A∧($$n_2n_1$＋X)→A	绝对X变址
AND $$n_2n_1$,Y	A∧($$n_2n_1$＋Y)→A	绝对Y变址
AND ($n,X)	A∧(有效地址)→A	先变址(X)间接寻址
AND ($n),Y	A∧(有效地址)→A	后变址(Y)间接寻址

AND 指令影响标志位N和Z。

2．ORA——逻辑"或"指令

ORA的作用是将累加器内容同M的内容相"或"，结果送累加器，即$A \vee M \to A$。ORA指令具有与AND指令相同的八种寻址方式，执行ORA指令也只影响标志位N和Z。

3．EOR——逻辑"异或"指令

这条指令的作用是将累加器内容与M的内容相"异或"，结果送累加器，即$A \veebar M \to A$。

逻辑运算操作是各位独立进行的，彼此间不存在借位、进位关系，它们都不影响进位标志C。

AND指令常用于屏蔽某些位或者取出某些位；OR指令常用于使某些位置1，或求混合信息（逻辑加）；EOR指令可用于求反码，6502 CPU没有求反码指令；BIT指令用于位检测，常与条件转移指令配合使用。

例6．已知＄6000单元中的内容为＄AD，要求取出它的低四位，并将其存入＄6001单元中。

解：实现要求的源程序如下：

	M	
6000	10101101	AD
6001	00001101	0D

```
LDA   $6000   ;   $(6000)→A
AND   $0F     ;   A∧0F→A
STA   $6001   ;   A→($6001)
BRK           ;   结束
```

例7．若要取出例6＄6000单元中的D_5位存A，则可用以下指令实现：

```
⋮
LDA   $6000   ;   ($6000)→A
AND   $20     ;   A∧$20→A
```

⋮

由上二例可见，欲取出某位，可用一个该位为1，其余位为0的数同A中内容相"与"即得。比如上例中要取出D_5位，就用一个D_5位为1，其余位为0的数，即$(00100000)_2$ = $ 20，与A中内容相"与"即得。

例8．将$6000单元中的内容求反后送$6001单元。

	M	
6000	10101101	AD
6001	01010010	$\overline{AD}$

```
LDA   $6000   ;   $6000→A
EOR   #$FF    ;   A⊻$FF→A
STA   $6001   ;   A→($6001)
BRK           ;   结束
```

执行上述程序后，$6001单元中即为$AD的反码$52。

例9．若要使上图中$6000单元的高四位皆为1，低四位不变，可用以下指令实现：

```
LDA   $6000   ;   ($6000)→A
ORA   $F0     ;   A∨$F0→A
STA   $6000   ;   A→($6000)
```

上述指令执行后，$6000单元的内容就由10101101变成11111101，所以利用ORA指令可以置某些位为1。

3.2.6 移位指令

1．ASL——算术左移指令

其功能是将M中各位依次向左移一位，最高位移入标志位C中，最低位补0，如图3.11所示。ASL指令相当于把M中内容乘以2。当M中的内容是带符号数补码时，其真值不能超过±64，否则产生溢出。

M代表累加器A或某一存储单元。ASL指令有五种寻址方式，其格式见表3.12。

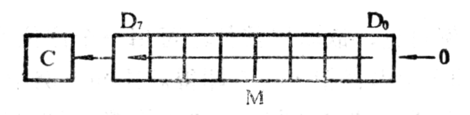


图3.11　ASL指令功能示意图

表3.12　ASL指令寻址方式

指令格式	功　　能	寻址方式
ASL　A	累加器A中内容算术左移一位	累加器寻址
ASL　$n	零页$n单元中内容算术左移一位	零页寻址
ASL　n_2n_1	n_2n_1单元中内容算术左移一位	绝对寻址
ASL　$n，X	$n+X单元中内容算术左移一位	零页X变址
ASL　n_2n_1，X	n_2n_1+X单元中内容算术左移一位	绝对X变址

2．LSR——逻辑右移指令

其功能是将M（含义与图3.11相同）中各位依次右移一位，最低位D_0移入进位位C中，最高位补0，如图3.12所示。

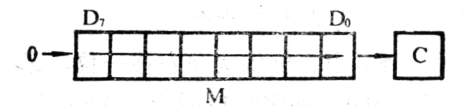


图3.12　LSR指令功能示意图

LSR指令对于M中的无符号数或正数相当于除以2，对于负数则不能（因为最高位补0，会把负数变成正数，导致结果出错）。执行LSR指令后，M中内容为其原内容除以2

后所得商的整数部分，小数存放在C中。LSR指令具有同ASL指令相同的五种寻址方式。

LSR指令和ASL指令均影响标志位C，N，Z。

3. ROL——循环左移指令

其功能是将M中的内容连同进位标志C一起依次循环左移一位，如图3.13所示。图中的含义与图3.11同。

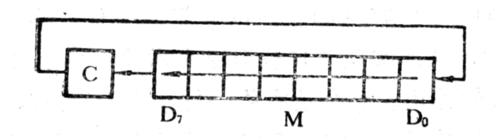


图3.13 ROL指令功能示意图

ROL指令与ASL指令具有相同的五种寻址方式。

4. ROR——循环右移指令

其功能是将M中内容连同进位标志C一起依次循环右移一位，如图3.14所示。

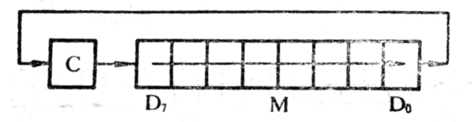


图3.14 ROR指令功能示意图

ROR指令和ROL指令具有相同的五种寻址方式，且执行结果都要影响标志位C，N，Z。

例10．将一个16位无符号数＄C7BD除以2（假设数存在＄6001和＄6000单元中，高位字节存高位地址，低位字节存低位地址）。

解：只要用以下两条指令将16位无符号数整个右移一位

就可以实现题目的要求。

```
LSR     $6001
ROR     $6000
```

这两条指令的联合移位情况如图 3.15 所示。由图可见，执行这两条指令后，存于＄6001和＄6000单元中的16位无符号数依次向右移了一位，相当于除以2。原高位字节的 D_0 位是首先通过LSR指令移入C中，然后再通过循环移位指令ROR再将C中内容（即原高位字节的D_0位）移到低位字节的D_7位中，从而实现了16位数整个右移一位，进位位C在此起了连接的作用。整个数右移一位相当于除以2，其商的整数部分存原单元＄6001和＄6000单元中，小数存进位位C中。指令执行后，＄6001，＄6000单元和进位位C中的内容＄63DE.8恰为＄C7BD除以2所得的商。

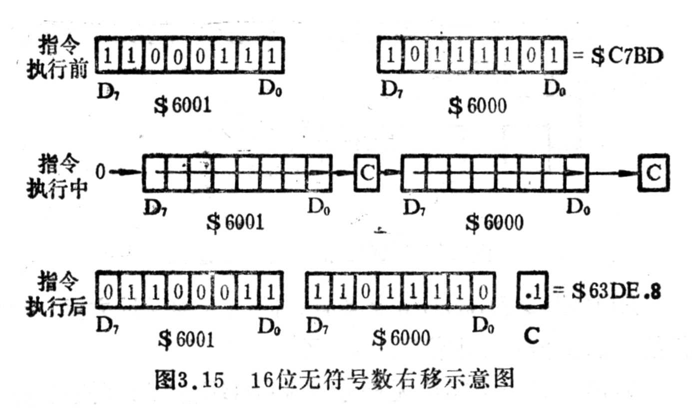


图3.15　16位无符号数右移示意图

例11．将＄3000单元内容拆成两段，每段四位，并将高四位存入＄3002单元，低四位存入＄3001单元，如图3.16所示。

解：先将＄3000单元的高四位屏蔽，取出低四位后送＄3001单元；然后再将＄3000单元中的低四位屏蔽，取出高四位，并逻辑右移四次后存入＄3002单元（或将取出的高四位先存入＄3002单元后，再逻辑右移四位也可以）。实现题意的源程序如下：

	M	
3000	X	Y
3001	0	Y
3002	0	X
	⋮	

图 3.16

```
LDA   $3000   ;  ($3000)→A，取操作数到A
AND   #$0F    ;  屏蔽高四位，取出低四位
STA   $3001   ;  将取出的低四位送$3001单元
LDA   $3000   ;  再取原操作数到A
AND   #$F0    ;  屏蔽低四位，取出高四位
LSR   A       ;
LSR   A       ;
LSR   A       ;
LSR   A       ;  逻辑右移四次，将原高四位挪到低四位
STA   $3002   ;  存原高四位于$3002中
BRK           ;  结束
```

3.2.7 堆栈操作指令

所谓堆栈是计算机在RAM中开辟的一个特殊的存储区，在这个存储区中必须遵循“先进后出”或者说“后进先出”的存（进）数、取（出）数规则，这个特殊的存储区域就称为堆栈。就象码头上的货栈一样，放货时需要从底往上放，取货时则必须先取上面的（后放的），后取下面的（先放的），这就是“先进后出”或“后进先出”规则。

在堆栈中存放数据是从地址码高的单元到地址码低的单元依次存放的，最后存放数据的那个单元称为栈顶，栈的另一端则称为栈底，见图3.17。栈底是堆栈中的最高地址码单元，栈顶则是当前堆栈的最低地址码单元。为了自动指出栈顶的位，6502 CPU把邻近栈顶的一个空单元的地址码存入寄存器S（称为堆栈指针）。一个数据进栈后，栈顶地址码减1，栈指针S自动减1；从堆栈取出一个数据后，栈指针自动加1。所以，栈顶的位置是随数据的进栈和出栈而浮动的，其当前位置由堆栈指针S反映出来。

由于6502规定它的堆栈只能设置在第1页中，即只能把$0100～$01FF这256个单元作为堆栈区域，所以栈指针S是一个八位寄存器，它指示当前栈顶空单元地址码的低八位，高八位固定为01。

图3.17表示向堆栈依次存入N_1，N_2和N_3三个数，然后再取出N_3，N_2两个数的操作过程以及堆栈的变化情况。

1. PHA——累加器进栈指令

指令功能是$A \rightarrow M_S$，$S-1 \rightarrow S$。即把累加器A的内容送入栈指针S所指的栈顶空单元M_S中，然后栈指针S自动减1。PHA指令不影响P的状态。

2. PHP——标志寄存器进栈指令

指令功能是$P \rightarrow M_S$，$S-1 \rightarrow S$。即把标志寄存器P的内容送入栈指针S所指的栈顶空单元M_S中，然后S自动减1。

3. PLA——累加器出栈指令

指令功能是$S+1 \rightarrow S$，$M_S \rightarrow A$。即首先使栈指针S加1，得到栈顶的地址码，然后将此地址单元中的内容取出送累加器A。执行PLA指令要影响标志寄存器P中的N，Z位。

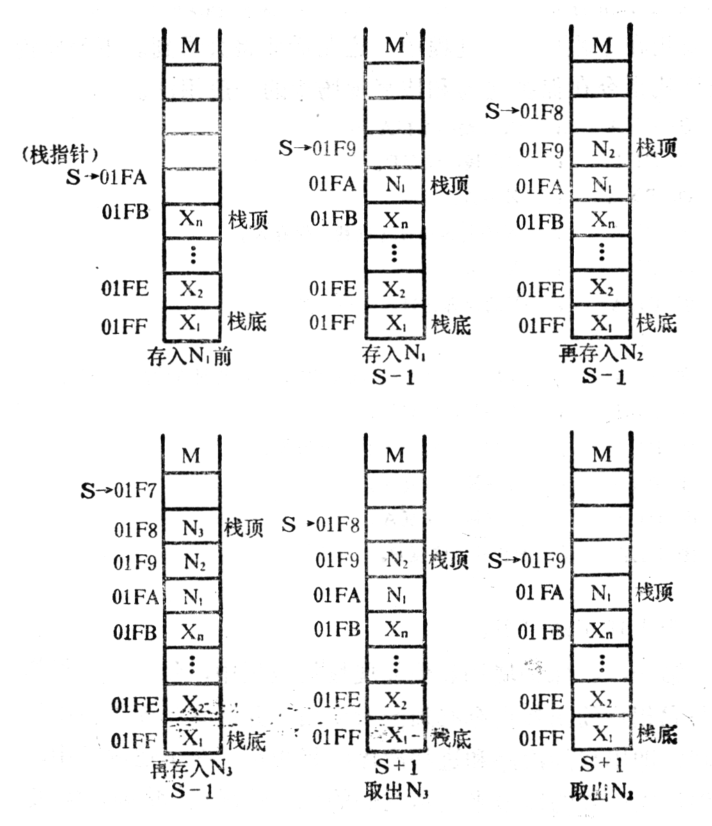


图3.17 堆栈操作示意图

4．PLP——标志寄存器出栈指令

功能是S＋1→S，M_S→P。即首先S自动加1，得到栈顶地址，然后从栈顶中取出内容送P。

进栈指令PHA，PHP，主要用于保护现场(保存CPU各寄存器的状态)；出栈指令PLA，PLP主要用于恢复现

场（恢复寄存器A、P原来的状态）。保护现场和恢复现场主要用在子程序调用过程中，这在后面将会看到。下面将说明堆栈指令在保护现场和恢复现场中的一般用法。

① PHA ； A进栈，保存A。
② PHP ； P进栈，保存P。
③ TXA ； X→A。
④ PHA ； X的内容通过A进栈，保存X。
⑤ TYA ； Y→A。
⑥ PHA ； Y的内容通过A进栈，保存Y。
⋮
⑥ PLA ； A出栈。根据"后进先出"原则，此时A中内容为Y。
⑤ TAY ； A→Y，恢复Y。
④ PLA ； A出栈。此时A中内容为X。
③ TAX ； A→X，恢复X。
② PLA ； A出栈，恢复A。
① PLP ； P出栈，恢复P。
⋮

由上述用法可见：

第一，在保护现场和恢复现场的应用中，每一条进栈指令与它相应的出栈指令必须配对使用，并遵守"先进后出"原则，即出栈的次序和进栈次序刚好相反（在上述例子中，我们用①…⑥的编号来指示它们的配对情况）。否则，恢复现场的操作将会出现"张冠李戴"的现象。

第二，只有寄存器A和P才有堆栈指令；寄存器X，Y没有堆栈指令，它们的进栈和出栈操作只能通过累加寄存器A进行。如上述③和④指令联合使用，可实现X进栈，⑤和⑥指令联合使用，实现Y进栈。出栈也必须如此配合使用，但次序刚好相反。

3.2.8 转移指令

1. JMP——无条件转移指令

执行这条指令，CPU转移到操作数所指定的地址去执行指令，它有两种寻址方式：

① JMP $\$n_2n_1$ ； $\$n_2n_1 \rightarrow PC$，CPU转向$\$n_2n_1$单元执行指令。绝对寻址方式。

② JMP $(\$n_2n_1)$ ； 将$\$n_2n_1$和$\$n_2n_1+1$单元的内容取出来送PC，即$(\$n_2n_1) \rightarrow PC_L$，$(\$n_2n_1+1) \rightarrow PC_H$，CPU随即转向PC指示的单元去执行指令。

2. 条件转移指令

① BEQ $n——零转移，即标志位Z＝1，则转移，否则继续（顺序执行程序）。所有条件转移指令都是相对寻址方式，$n表示相对转移量，以下同。例如：

```
指令存放地址        指令
   ⋮                ⋮
$0300      LDA $06    ;  ($06)→A。
$0302      BEQ $06    ;  若Z=1，则CPU转到$0304+$06去
                         执行SBC指令。
$0304      AND #$0F   ;  若Z=0，则顺序执行AND指令。
   ⋮          ⋮
$030A      SBC $1000
   ⋮          ⋮
```

② BNE $n——非零转移，即Z＝0时转移，否则顺序执行程序。指令执行情况与BEQ类似，只是转移条件（判据）不同而已。BEQ是零转移（Z＝1转移）；BNE是非零转移（Z＝0转移）。

③ BCC $n——无进位转移，即标志位C＝0时转移，

否则继续。

④ BCS ＄n——有进位转移，即C＝1转移，否则继续。

⑤ BPL ＄n——正转移，即标志位N＝0转移，否则继续。

⑥ BMI ＄n——负转移，即N＝1转移，否则继续。

⑦ BVC ＄n——标志位V＝0转移，否则继续。

⑧ BVS ＄n——标志位V＝1转移，否则继续。

以上八种条件转移指令皆为相对寻址方式的二字节指令，第一字节为操作码，第二字节为相对偏移量，它们都不影响标志寄存器P。

3.2.9 转子指令和返回指令

某些程序，如乘法程序，除法程序，数码转换程序等等，在不同的应用程序中可能多次使用；如果每次都重新编写，将会浪费许多人力，并占用较多的存储单元。为此，把一些常用的程序编制成独立的程序段，放在内存中供大家使用。这些独立的程序段称为子程序。有了子程序之后，在编写某些程序时，可以根据需要转向相应的子程序；执行子程序以后，再返回到原来的程序继续执行。

转向子程序称为子程序调用，是通过执行转子指令（或称子程序调用指令）来实现的。调用子程序的程序称为主程序，子程序执行完以后一般是通过执行子程序返回指令，返回到主程序。返回时是返回到转子指令的下一条指令，继续执行主程序。上述过程如图3.18所示。

1. JSR——转子指令

执行该指令后，CPU转移到该指令操作数所指出的子

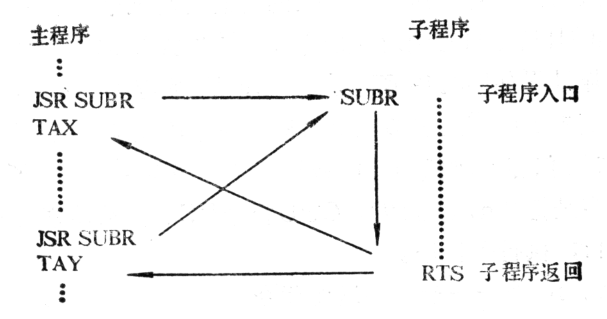


图3.18 子程序调用和返回过程示意图

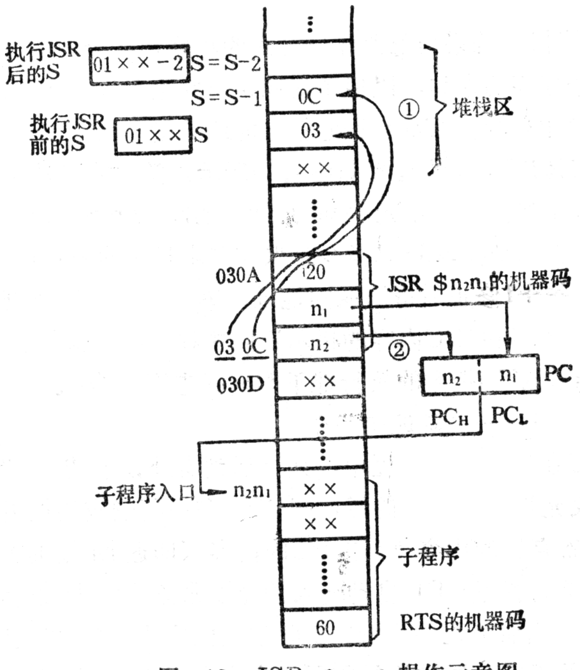


图3.19 JSR $ n_2n_1操作示意图

程序入口去执行子程序。JSR的机器码有三个字节。

JSR指令只有绝对寻址方式，其格式为JSR $\$n_2n_1$，其操作过程如图3.19所示。第一步是保存断点，将JSR $\$n_2n_1$机器码第三字节在存储器中的存放地址（图3.19中为$030C)保存到堆栈中，以备执行子程序返回指令时使用；第二步是把$\$n_2n_1$送PC，CPU根据PC的指示转移到子程序入口，执行子程序。

2．RTS——子程序返回指令

其功能是使CPU返回到主程序断点处继续执行主程序。

RTS指令是采用隐含寻址方式的单字节指令，它通常是用于子程序的末尾，以便子程序执行完之后返回主程序，见图3.18和3.20。RTS指令通常是同JSR指令联合使用的。

执行RTS指令的过程如图3.20所示。第一步是从堆栈中连续取出栈顶两个单元（如图3.20中的S+1和S+2单元）的内容，它们是执行JSR指令时保存到堆栈中的断点地址，将此地址作+1修正后得出返回地址；第二步是把返回地址送PC，CPU根据PC内容的指示返回主程序继续执行。

JSR和RTS指令的执行都不影响标志寄存器P。

3．RTI——中断返回指令

CPU在执行程序过程中，如果遇到有外设申请中断，而CPU又允许响应中断的话，则CPU在中断响应周期中，首先把P寄存器的内容及当前PC的内容（即返回地址）送进堆栈保存，然后CPU自动转入为外设服务的中断服务程序去执行；当服务程序执行完毕时就返回主程序，继续执行。这里的返回是通过执行RTI指令来实现的，RTI指令总

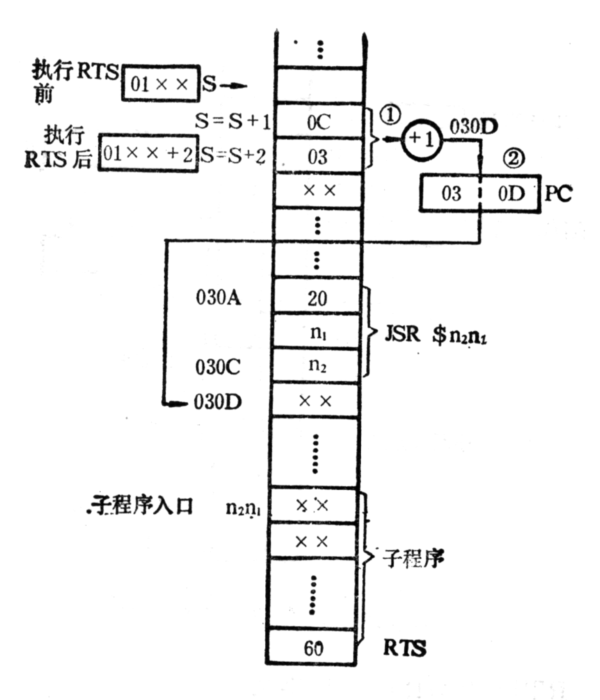


图3.20 RTS指令操作示意图

是放在中断服务程序的末尾。

中断服务程序也可以看作是一种子程序，但它不是通过执行转子程序转入，而是CPU响应中断后自动转入，而且服务程序的末尾不用RTS指令，而是用RTI指令。

RTI指令是采用隐含寻址方式的一字节指令，它只能用在中断服务程序的末尾。执行RTI指令的过程，首先是把保存在堆栈中的P寄存器的状态送回P，并把返回地址送回PC，然后CPU由PC内容指示的地址返回主程序去执行，如图3.21所示。

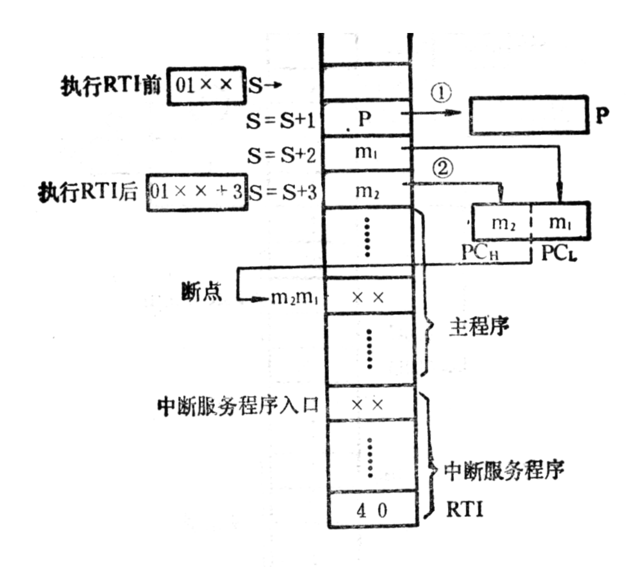


图3.21 RTI指令功能示意图

RTI指令与RTS的区别是：

① RTI只能用在中断服务程序的末尾，RTS指令用在子程序末尾；

② RTI指令比RTS多一个P寄存器的出栈操作，RTS指令比RTI多一个对出栈地址做加1修正的操作。可见，它们的功能彼此不尽相同，故不能互相代替，编写程序时切不能用错了。

3.2.10 其它指令

1. BRK——软件中断指令

这是采用隐含寻址方式的单字节指令。可以用该指令在

程序中设置断点，CPU执行到BRK指令时就停止用户程序的执行，返回监控程序，屏幕上显示出CEC-I机监控状态的标志符*。此时，用户可用监控命令检查程序的执行结果。

由于BRK指令可以命令CPU中断用户程序的执行，故称它为软件中断指令。执行BRK指令对P寄存器的影响是将标志位B置1，I置1，其余各标志位不变。B置1表示此时CPU处于软件中断状态，因而I也必须置1，以表示此时CPU禁止对外设中断申请的响应。

2. 空操作指令NOP

NOP是采用隐含寻址方式的单字节指令，执行时间是两个时钟周期。CPU执行NOP指令，除了使PC+1，并占用了两个时钟周期外，不做其它任何操作，故称空操作指令。

NOP指令可用于延时程序中，也常用于程序调试中。比如，调试程序时，由于某种原因而需要暂时删除某条指令，将来统调时还要将这条指令添上，就常在该指令位置用NOP指令暂时替代，所用NOP指令的个数应和待删除指令的字节数相同，将来统调时又把NOP指令取消，再添上需要的指令。这样就不致于引起程序存储区地址的变化，从而避免了调试中由于地址变化而造成的混乱和错误。

3.3 6502指令操作小结

现在，我们把6502各种指令的操作功能和对标志寄存器各位的影响列于表3.13，供查阅。

表3.13 6502指令系统简表

助记符	功 能	操 作	要影响的标志位
			N V · B D I Z C
ADC	带进位加	A + M + C → A	N V · · · · Z C
AND	逻辑与	A ∧ M → A	N · · · · · Z ·
ASL	算术左移一位	C ← [D_7 ←— D_0] ← 0	N · · · · · Z C
BCC	进位为零跳转	C = 0跳转	· · · · · · · ·
BCS	进位为1跳转	C = 1跳转	· · · · · · · ·
BEQ	结果为零跳转	Z = 1跳转	· · · · · · · ·
BIT	位测试	A ∧ M	N V · · · · Z ·
BMI	结果为负跳转	N = 1跳转	· · · · · · · ·
BNE	结果非零跳转	Z = 0跳转	· · · · · · · ·
BPL	结果为正跳转	N = 0跳转	· · · · · · · ·
BRK	软件中断	强迫中断	· · · 1 · 1 · ·
BVC	溢出位为零跳转	V = 0跳转	· · · · · · · ·
BVS	溢出位为 1 跳转	V = 1跳转	· · · · · · · ·
CLC	进位位清零	0→C	· · · · · · · 0
CLD	十进制标志清零	0→D	· · · · 0 · · ·
CLI	禁止中断标志位清零	0→I	· · · · · 0 · ·

续表

助记符	功　能	操　作	要影响的标志位							
			N	V	·	B	D	I	Z	C
CLV	溢出位清零	$0 \to V$	·	0	·	·	·	·	·	·
CMP	比　较	$A - M$	N	·	·	·	·	·	Z	C
CPX	X比较	$X - M$	N	·	·	·	·	·	Z	C
CPY	Y比较	$Y - M$	N	·	·	·	·	·	Z	C
DEC	减 1	$M - 1 \to M$	N	·	·	·	·	·	Z	·
DEX	X减 1	$X - 1 \to X$	N	·	·	·	·	·	Z	·
DEY	Y减 1	$Y - 1 \to Y$	N	·	·	·	·	·	Z	·
EOR	逻辑异或	$A \forall M \to A$	N	·	·	·	·	·	Z	·
INC	加 1	$M + 1 \to M$	N	·	·	·	·	·	Z	·
INX	X加 1	$X + 1 \to X$	N	·	·	·	·	·	Z	·
INY	Y加 1	$Y + 1 \to Y$	N	·	·	·	·	·	Z	·
JMP	无条件跳转	跳转到新地址	·	·	·	·	·	·	·	·
JSR	转　子	跳转到子程序	·	·	·	·	·	·	·	·
LDA	取数送A	$M \to A$	N		·	·	·	·	Z	·
LDX	取数送X	$M \to X$	N	·	·	·	·	·	Z	·
LDY	取数送Y	$M \to Y$	N	·	·	·	·	·	Z	·
LSR	逻辑右移一位	$0 \to [D_7 \to D_0] \to C$	0	·	·	·	·	·	Z	C

助记符	功　能	操　作	要影响的标志位 N V · B D I Z C
NOP	空操作		· · · · · · · ·
ORA	逻辑或	A∨M→A	N · · · · · Z ·
PHA	A进栈	A→Ms，S－1→S	· · · · · · · ·
PHP	P进栈	P→Ms，S－1→S	· · · · · · · ·
PLA	A出栈	S＋1→S，Ms→A	N · · · · · Z ·
PLP	P出栈	S＋1→S，Ms→P	恢复各标志位原来状态
ROL	循环左移一位	[D_7 ← D_0] ← [C] ← (循环)	N · · · · · Z C
ROR	循环右移一位	[D_7 → D_0] → [C] → (循环)	N · · · · · Z C
RTI	中断返回	中断返回	恢复各标志位原来状态
RTS	子程序返回	子程序返回	· · · · · · · ·
SBC	带进位减法	$A-M-\overline{C}\rightarrow A$	N V · · · · Z C
SEC	进位位置1	1→C	· · · · · · · 1
SED	十进制标志置1	1→D	· · · · 1 · · ·
SEI	禁止中断位置1	1→I	· · · · · 1 · ·
STA	A送存	A→M	· · · · · · · ·
STX	X送存	X→M	· · · · · · · ·

续表

助记符	功　能	操　作	要影响的标志位
			N V · B D I Z C
STY	Y送存	Y→M	· · · · · · · ·
TAX	A送X	A→X	N · · · · · · Z ·
TAY	A送Y	A→Y	N · · · · · · Z ·
TSX	S送X	S→X	N · · · · · · Z ·
TXA	X送A	X→A	N · · · · · · Z ·
TXS	X送S	X→S	· · · · · · · ·
TYA	Y送A	Y→A	N · · · · · · Z ·

表3.13用所符号说明：

① M_S表示堆栈指针S所指示的存储单元；

② 在"要影响的标志位"一栏中：

"·"表示没有影响。

"1"表示置1。

"0"表示置0。

"N""V""Z""C"表示要根据指令执行结果确定这些位是"0"还是"1"。

习　题

1. 6502 CPU有多少种寻址方式？请解释先变址(X)间接寻址和后变址（Y）间接寻址方式，并画图说明之。

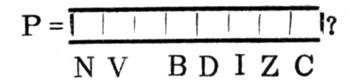
2. 请指出下列各条指令的寻址方式：

```
AND   #$n      ;   DEC   $n, X      ;
AND   $n       ;   DEC   $n2n1, X   ;
ADC   $n2n1    ;   DEX              ;
ADC   $n, X    ;   RTS              ;
ADC   ($n, X)  ;   CLC              ;
ADC   ($n), Y  ;   BEQ   $n         ;
```

3. 指出以下各指令中的错误，并按其意图进行改正。

```
LDA   ($0300)    ;   PHX
STA   $06, Y     ;   PLY
STX   $08, X     ;   LDS   $0300      ;
STY   $08, Y     ;   STY   ($6000)    ;
INC   $0300, Y   ;   JMP   $6000, X   ;
DEC   $0300, Y   ;   LD($6000), #$38
```

4. 请指出执行下面指令后，A = ?，




```
SEC
LDA   #$3A
ADC   #$3A
SBC   #$40
```

5. 设初始时（$6000）= $38，（$6001）= $7B（$6002）= $52，试述执行下面程序后，上述内存单元及标志寄存器P的内容。

```
CLC
LDA   $6000
ADC   $6001
STA   $6002
BRK
```

6．假设（$6001）=$35，（$6000）=$AE，试写出用移位指令实现$35AE除以2的程序段，并写出运算结果。

7．如果（$03FF）=$C0，（$03FE）=$30，请指出执行下面指令后，PC=？

```
① $6000 : JMP   $6100
② $6000 : JMP   ($03FE)
③ $6000 : JSR   $FF3A
④ $6000 : BMQ   $08
```

8．请写出下面指令的机器码，并说明执行它们之后PC=？S=？堆栈内容如何变化？

```
① $6000 : LDX   #$BF
          TXS
          JSR   $0300
② $6000 : LDX   #$EE
          TXS
          CLC
          LDA   #$0A
          SBC   #$0B
          BCC   $EF
          JSR   $FB09
```

第四章　程 序 设 计

汇编语言是一种与 CPU 的内部结构及其指令系统密切相关的程序设计语言。在前面两章中已经介绍了 6502 CPU 的内部结构，它的指令系统及寻址方式。本章将继续介绍 6502 CPU 的汇编语言的语句格式，伪指令及汇编语言程序设计的基本方法。

用汇编语言编写的程序称为汇编语言源程序，它要转换成目标程序——用机器码表示的程序，计算机才能识别和执行。完成这种转换工作的软件称为汇编程序，这个转换过程称为汇编。

4.1　汇编语言的语句格式

6502的汇编语言包括两类语句：一类是符号指令语句，另一类是伪指令语句。符号指令语句即6502 CPU指令系统中的符号指令，伪指令语句将在下一节中介绍。

典型的汇编语句由以下四部分组成：

标号　　操作码　操作数 ；注释

前面各部分之间用空格作为分隔符，操作数和注释之间用分号"；"分隔。在这四部分中，操作码和操作数是必不可少的部分，但标号和注释可根据需要选用，并不是每条语句都必须有。

1. 标　号

标号是指令的符号地址。源程序被汇编成目标程序时，应赋予标号以绝对地址，就是指令的机器码在内存中存放的实际地址。用标号表示地址便于程序的修改和转移指令的书写，如子程序的入口地址，转移的目标地址等都常用标号表示。比如：

```
标号      操作码    操作数；  注释
START     LDA      $1000；  将$1000单元的内容送A
          STA      $2000；  将A中内容送$2000单元
```

上述第一条语句用了标号START，表示该语句指令的机器码应从START代表的内存单元开始存放。如果START代表$0300，则第一条语句指令的机器码就从$0300单元开始存放。第二条语句没有标号，它的机器码接续上面的内存单元存放。同一程序中出现的相同标号代表同一地址码，除非重新给它赋值。

使用标号要注意以下几点：

① 标号必须以英文字母开头，并不得多于八个字符，以空格作结束符。

② 不能用指令的助记符作标号。比如不能用LDA，CMP，DEC等等作标号。

③ 也不能用6502 CPU的寄存器名：A，X，Y，S，P作标号。

2．操作码

操作码由指令助记符组成，它是每一条语句必须有的字段，如LDA，STA等。

3．操作数

在汇编语句的操作数部分，可以是操作数自身，也可以是操作数所在地址，或所在地址的地址，这要由各指令的寻

址方式决定。在汇编语句中，操作数可以用以下形式给出：

① 标号，如下面例1第七条语句中的LOOP。

② 常数，可以用任一进制数。如例1中第1，3，4条语句中用的是16进制数，第6条语句中用的是十进制数，其它还可以用8进制数，字符串常数等。

例1．数据块传送。

将存储在$1000单元开始的100个数传送到自$1100单元开始的内存区域中。源程序如下：

```
        ORG   $300       ；以下程序的目标程序从$300单元开始
                           存放
        LDX   #00        ；0→X
LOOP    LDA   $1000,X    ；($1000+X)→A
        STA   $1100,X    ；A→($1100+X)
        INX              ；X+1→X
        CPX   #100       ；X-100
        BNE   LOOP       ；Z=0，未送完，转LOOP
        BRK              ；结束
```

CEC-Ⅰ机规定：十进制常数不用标志；十六进制数必须在数前加$作标志；八进制数前加@作标志，如@36表示八进制数36。

字符常数用单引号来标志，如LDX #'A，表示把字符A的ASCII码$C1送X。字符串常数则要在字符串的首尾都加上单引号，如：伪指令

```
DFB 'ABC' ；
```

表示把字符串A，B，C的ASCII码依次存放到由DFB所指定的单元开始的连续三个单元中去。

③ 表达式，操作数也可以用表达式来表示，表达式由运算符+、-、*、/、>、<等构成。如：LDA NEXT

－1中，使用了标号和常数组成的表达式。它表示把标号NEXT对应的地址码减1作为绝对地址，再从这个地址中取出操作数送A。运算符“<”表示选择高字节；“>”表示选择低字节，例如：

```
LDA   #<$1050      ；将立即数$1050的高字节$10送A
LDX   #>$1050      ；将立即数$1050的低字节$50送X
```

当表达式中有多个运算符时，运算顺序是从左到右，即所有运算符具有同等的优先级，运算时从左到右依次进行。如表达式15＋<2000－2的值是：先将15加上2000的高字节，然后再减去2。

4. 注　释

注释的作用是对所用的指令给出说明，以便于对程序的阅读和理解。注释部分是可选部分，不是每条语句都必须有。

4.2 伪　指　令

伪指令与符号指令的区别是：它没有对应的机器码，不能引起计算机硬件的某种操作，所以称为“伪”指令。

伪指令的作用仅仅是向汇编程序提供某些信息，帮助汇编程序完成从源程序到目标程序的翻译工作。比如，为目标程序分配一定的存储区，给标号赋值，把给定数据存入指定存储单元等等。具有相同型号CPU的微型机，具有相同的符号指令系统，但其伪指令则可能因采用的汇编程序不同而不尽相同。下面介绍的是CEC-Ⅰ机常用的汇编程序使用的伪指令。

1. ORG——起始指令

格式：ORG $ n_2n_1；其作用是告诉汇编程序，在ORG指令之后的程序或数据从$ n_2n_1所指定的内存单元开始存放。如例1中的ORG $300，表示它后面的源程序翻译成目标程序后，目标程序要从$300单元开始依次存放。

2. EQU——等值指令（或称赋值指令）

格式：标号　EQU　操作数

其功能为令标号＝操作数，如：

```
          ORG   $300      ；以下程序的机器码从$300开始存放
COUNT     EQU   $F634     ；COUNT＝$F634
          LDA   #10       ；10→A
          STA   $06       ；A→($06)
          JSR   COUNT     ；转到地址$F634去执行程序
           ⋮
```

如果该程序中其它地方也用了标号COUNT，则它都代表$F634，除非用EQU指令给它重新赋值。

3. DFB——定义字节指令

格式：标号　DFB　操作数

其功能是告诉汇编程序，由DFB定义的操作数应从标号指明的地址开始存放。如：

```
FIRST DFB 1, 2, 3, 5;
```

表示把数1,2,3,5依次存放到从FIRST指定的内存单元开始的存储区，每个数占一个字节。

```
NUMBER DFB 'A, 'B, 'C, 'D;
```

表示把字符A，B，C，D的ASCII码依次存放到内存中从NUMBER开始的存储单元。由此可见，当操作数是单个字符时，必须将字符括在单引号内，而且各字符或单字节数之

间要用逗号隔开。

上述标号FIRST，NUMBER所指单元的绝对地址究竟是多少？这由两种方法来确定。一种是由ORG指令定义；另一种是接续前面的地址。比如：

```
        ORG  $300
FIRST   DFB  1, 2, 3, 5          ；表示FIRST = $300
NUMBER  DFB  'A, 'B, 'C, 'D；表示NUMBER = $304
```

此外，标号也可由EQU赋值，以下同。

记住，DFB只定义单字节数。

4. DW——定义字指令

格式：　标号　DW　操作数

其功能是告诉汇编程序，将DW定义的操作数（均为双字节数）从标号指定的单元开始存放，每一个数存放两个单元，数的高八位存放在高地址单元，低八位存放在低地址单元。当操作数字段有多个操作数时要用逗号隔开。

DW指令是定义双字节数，所以用DW指令存放地址十分方便。例如：

```
        ORG   $06                ；ADDRES = $06
ADDRES  DW    $AF04, $AFB6       ；从零页$06单元开始连
                                   续存放两个地址码，见
                                   图4.1。
          ⋮
        JMP   (ADDRES)           ；转移到 $AF04去执
                                   行指令。
          ⋮
        JMP   (ADDRES+2)         ；转移到 $AFB6去执
                                   行指令。
          ⋮
```

5. DDB——定义双字节指令

格式： 标号 DDB 操作数

DDB指令的功能与DW指令的相同之处是定义双字节数，不同之处是存放双字节数的顺序不同。DDB是把它定义的双字节数的高八位存低地址单元，低八位存高地址单元，这个顺序与DW正相反。例如：

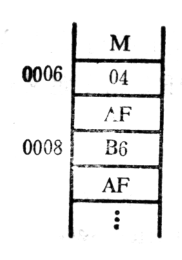


图4.1 DW指令定义双字节数的存储情况

```
        ORG   $06
ADDRES  DDB   $AF04;
```

表示$AF存零页中$06单元，$04存$07单元，与图4.1中情形正好相反。

6. DS——定义存储器指令

格式： 标号 DS 操作数

其功能是告诉汇编程序，从标号所指定的单元开始预留操作数所指定个数的存储单元，用来存放中间结果或最后结果。例如：

```
BUFFER  DS  10;
```

表示从BUFFER单元开始，预留10个单元待用。

7. END——源程序结束指令

END指令的作用是告诉汇编程序，源程序在此结束，END后面的指令不再汇编。所以，END指令一定要放在源程序末尾。

以上是APPLE-Ⅱ汇编程序的常用伪指令，它适用于CEC-Ⅰ机。选用的汇编程序不同，其伪指令也不完全相

同，但大同小异。最好阅读它的使用说明后再选用您需要的伪指令。

4.3 汇编语言程序设计方法

任何一个程序，不论其简单还是复杂，都可以由三种基本结构组合而成。这三种基本结构是：顺序结构，分支结构和循环结构。当然，在此基础上还可以有一些辅助结构，但这些辅助结构仍可看成是上述基本结构的扩展或变化。下面，我们以这三种基本结构为基础来介绍CEC-Ⅰ机的汇编语言程序设计方法。

4.3.1 顺序结构程序

这是结构最简单的程序，CPU从程序的第一条指令开始依次顺序执行，没有任何跳转。或者说程序计数器PC的内容从程序的第一条指令的首地址开始，每取一字节指令代码，PC+1；再取一字节指令代码，PC再加1；顺序执行程序，直到程序结束。

例2. 求两个数之和。

```
ORG   $300
CLC              ；清进位标志C
CLD              ；清十进制标志D
LDA   $1000      ；($1000)→A
ADC   $1001      ；A+($1001)+C→A
STA   $1002      ；A→($1002)
BRK              ；结束
END
```

这是一个很简单的顺序结构的加法程序。它是把＄1000单元和＄1001单元的内容按二进制数相加，结果送到＄1002单元中存放（假设二数之和不溢出）。整个程序顺序执行，无任何跳转。

4.3.2 分支结构程序

具有分支结构的程序的最大特点是程序中必须含有判断语句。判断语句由条件转移指令来实现，根据程序执行中条件是否满足来决定程序是顺序执行还是跳过某些程序段执行，其典型的程序流程见图4.2。

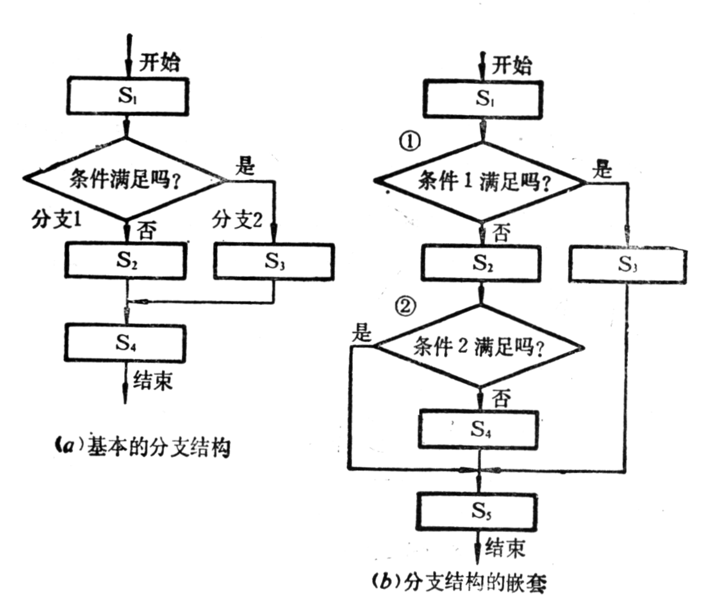


图4.2 分支结构程序流程图

图4.2(*a*)是一个基本的分支结构程序的流程图。图中S_1，S_2（或S_3），S_4框表示顺序执行的程序段，其中S_2和S_3是两个并行的分支，当判断框中的条件不满足（否）时，执行S_2程序段；当条件满足（是）时执行S_3程序段。图4.2（*b*）是分支结构允许嵌套的情况，判断框②嵌套于判断框①产生的分支中，或者说分支中还可产生分支，组成复杂的分支结构。

例3．比较两个无符号数的大小。

将分别存于$1000单元和$1001单元的两个无符号数进行比较，比较结果将大数存$1002单元。实现上述要求的程序流程见图4.3，源程序如下：

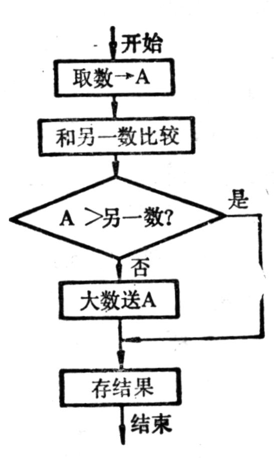


图4.3 比较两个无符号数大小的流程图

```
        ORG   $300
        LDA   $1000     ；($1000)→A
        CMP   $1001     ；A－($1001)
        BCS   NEXT      ；若C＝1，表明A中为大数，转到NEXT
        LDA   $1001     ；若C＝0，表明A中为小数，把大数送A，
                          即($1001)→A
NEXT    STA   $1002     ；A→($1002)，存大数
        BRK             ；结束
        END
```

4.3.3 循环程序

在顺序结构程序中，每条指令只能执行一次而且必须执

行一次；在分支结构程序中，根据条件不同会跳过一些指令，执行另一些指令，但每条指令最多只执行一次。然而，在处理实际问题中，有时需要多次处理同一个问题，这就应当采用具有循环结构的程序，使某些程序段重复执行多次。这样，可使程序简短，占用内存少。汇编语言中没有专门的循环语句，而是靠转移指令来实现循环的。

典型的循环程序包括以下四个部分：

① 循环准备。这部分的主要工作是对与程序有关的各寄存器进行初始化，包括设置地址指针，循环次数，有关寄存器清零，等等。

② 循环处理。这是循环程序的核心部分，它是需要重复执行的实质性操作部分，又称为循环体。

③ 循环修改与控制。这部分是对循环中用到的一些参数进行修改和控制，如修改地址指针，循环次数，控制循环是否结束等。

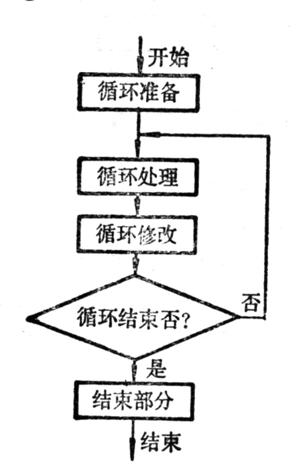


图4.4 循环程序流程图

④ 循环结束。跳出循环，分析和存放结果。

循环程序的典型流程如图4.4所示。对于一个具体的循环程序，这几部分有时不能截然分开。

在循环程序设计中，对循环次数的控制十分关键，搞不好就会陷入死循环。通常有两种办法控制循环：

① 循环次数已知时可用计数器控制循环；

② 循环次数未知时，则按问题必须满足的条件来控制循环。

例4. 求数据的累加和。

假设有一串无符号的单字节二进制数，这串数的长度存放在＄1000单元，而这串数则从＄1001单元开始存放。将这串数求和，并把和存入零页中＄06单元中（假设和不超过一字节，这样可以不考虑进位）。

实现上述要求的流程示于图4.5，源程序如下：

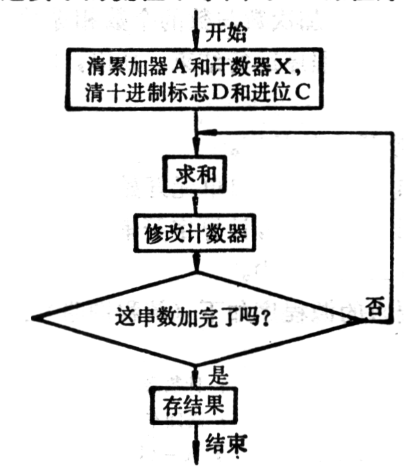


图4.5 例4流程图

```
ORG   $300
LDA   #00          ; 和初值=0
TAX                ; 计数器初值=0
CLD                ; 清十进制运算标志
CLC                ; 清进位标志
```

```
LOOP ADC  $1001,X     ; A+($1001+X)+C→A
     INX              ; X+1→X
     CPX  $1000       ; 数据累加完了吗?
     BNE  LOOP        ; 没有，继续循环
     STA  $06         ; 累加完了,存和
     BRK              ; 结束
     END
```

该程序中将字符串长度作分循环次数，它就是要参与求和的无符号数个数。程序中设置X作为计数器，同时X又作为 ADC 指令中用的变址寄存器。X 的初值设置为零，每加一次，X增1，当累加次数与数的个数相等时，累加完成，退出循环，存结果，程序运行结束。所以，本例是用数的个数来控制循环次数的。

例5．计算字符串的长度。

假设有一串字符从＄1001单元开始存放，字符串以回车符 CR 结束。请设计一程序，计算字符串长度(不算回车符)，并将结果存入＄1000单元中。

实现上述要求的源程序如下（流程图略）：

```
     ORG  $300
     LDX  #0          ; 计数器X清0
     LDA  #$8D        ; 送回车符CR的ASCII码$8D到A
LOOP CMP  $1001,X     ; A-($1001+X)
     BEQ  DONE        ; Z=1(若是回车符)，跳转到DONE，
                        循环结束
     INX              ; Z=0,则X+1→X
     JMP  LOOP        ; 循环
LONE STX  $1000       ; X→($1000)，存字符串长度
     BRK              ; 结束
     END
```

这个例子是按问题的条件（遇回车符就结束）来控制循环次数的。条件满足，循环结束，否则继续循环。另外，此循环程序中还包含了分支结构，由判断语句BEQ DONE 引起分支。

4.3.4 子程序

子程序也是由上述三种基本结构组成的，但其编制还有一些特殊要求需要处理。

为不同的目的而共用的一些处理，如代码转换，字符处理，乘除法运算及其它通用的函数计算等，都可以编制成相对独立的程序段，相对于主程序而言称它们为子程序。这样，主程序只要用转子指令就可调用子程序。采用主-子程序结构，有利于将大程序划分成若干个小的程序功能块，以便于编制和调试。由于子程序执行次数多，共享性大，因此，对子程序的编程设计要求高，程序应尽可能短，占用内存少，执行速度快。

在主-子程序结构中，由于主程序要用转子指令来调用子程序，因而在子程序的开头一定要给出入口地址，结束时不要忘了用子程序返回指令。

例6. 试把例5中计算字符串长度的程序改为主-子程序结构。

主程序

```
        ORG   $300
        LDA   #$8D
        JSR   SUBR
        STX   $1000
```

子程序

```
        ORG   $0C00
SUBR    LDX   #0
LOOP    CMP   $1001,X
        BEQ   DONE
```

```
BRK                                     INX
                                        JMP   LOOP
                              DONE      RTS
```

上述子程序只能计算从＄1001开始存放的字符串长度，通用性不强。如果把字符串存放的起始地址放在主程序中，则它就可以变成一个通用性强的子程序，更改以后的主、子程序如下：

```
        主程序                              子程序
        ORG   $300                          ORG   $C00
BUFFER  EQU   $1001                  SUBR   LDX   #0
NUMBER  EQU   $8D                    LOOP   CMP   BUFFER,X
        LDA   #NUMBER                       BEQ   DONE
        JSR   SUBR                          INX
        STX   BUFFER-1                      JMP   LOOP
        BRK                          DONE   RTS
```

这样，子程序SUBR（通常把子程序入口地址的标号作为子程序的名称）就是一个计算字符串长度的通用子程序。无论字符串从何单元开始，以什么标志结束，都可以调用它来计算，只要在主程序中把字符串起始地址通过伪指令 EQU 赋给标号BUFFER，把结束标志赋给NUMBER就成了。

从这个例子可以看出，为了便于共享子程序，还应对子程序有一个说明清单，它包括以下内容：

① 子程序名称。它常为子程序入口地址的标号。

② 子程序功能。比如上例子程序的功能是计算字符串长度。

② 入口条件。即进入子程序前，主程序需要提供的数据。比如，上例中的BUFFER，NUMBER应在主程序中

赋值。

④ 出口条件。即子程序结束时它处理的结果存放何处。比如，上例中计算出的字符串长度存X中，主程序从X中取结果。

⑤ 执行子程序时，要改变CPU中哪些寄存器的内容？比如，上例中要改变X寄存器，这样主程序在调用子程序之前，如果X的内容不需要保护就算了，如果需要保护，就可以用进栈指令保存起来，调用子程序结束之后再用出栈指令恢复X的内容。

子程序也可以嵌套，如图4.6所示。在需要时还可以构成多重子程序的嵌套，一般最多可嵌套八重。

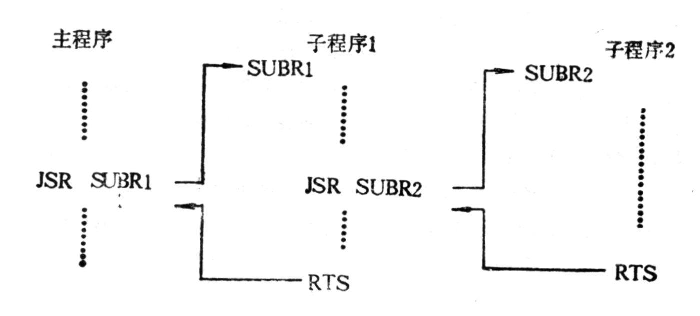


图4.6 子程序嵌套示意图

4.4 汇编语言实用程序

例7. 排序程序。

将从＄1001单元开始连续存放的一串无符号数按从小到大的升序重新排列，无符号数的个数N已存于＄1000单

元中。

排序的方法有许多种。下面，我们采用的基本方法是：从最末一个数开始，依次把它与数组中其余N-1个数相比较，每比较一次都把其中较大者放在最末一个单元，当与其余N－1个数比较完一遍以后，在最末一个单元中存放的就是数组中最大的数。接着进行第二轮比较，从次末一个单元开始，把它与其余N-2个数依次进行比较，每比较一次都把较大者放在次末一个单元中。这样，当第二轮比较完了之后，在次末单元中就是数组中次大的数。依次类推。当比较完N－1轮以后，这个数组就按升序重新排好了。实现这个排序方法的源程序如下：

```
         ORG   $300
         LDY   $1000      ; 取数的个数→Y
         STY   $06        ; 暂存$06单元
         DEC   $06        ; 数的个数减1，得比较总轮次
LOOP1    LDX   $06        ; 每轮比较次数→X
LOOP2    LDA   $1000,Y
         CMP   $1000,X    ; 两个数比较
         BCS   MAX        ; A中数大，不交换，转MAX
         PHA              ; A中数小，交换。A中应始终保持
                            大数
         LDA   $1000,X
         STA   $1000,Y
         PLA
         STA   $1000,X
MAX      DEX              ; X－1→X
         BNE   LOOP2      ; X≠0,表示一轮未完，继续比较
         LDY   $06        ; X＝0,则准备下一轮
         DEC   $06        ; ($06)－1→($06)，得下轮比较
```

```
                              总次数
        BNE   LOOP1          ；最后一轮吗?否，继续下一轮
        BRK                  ；是，结束
```

该程序设计巧妙之处是：利用零页中＄06单元的内容既控制了比较轮次，又通过X控制了每轮比较的次数。

例7是实现无符号数排序，因此排序中要进行无符号数大小的比较。比较两个无符号数的大小只须用标志位C便可判断，判断方法是：N_1-N_2（假设N_1，N_2为两个无符号数）时，若进位标志C＝1（即借位$\overline{C}=0$）时，则$N_1 \geqslant N_2$；若C＝0（即借位$\overline{C}=1$）时，则$N_1<N_2$。

当比较两个有符号数大小时，就不能用C标志判断了。比如当$N_1=-1$，$N_2=+2$，由于微机中带符号数都是用补码表示的，$[N_1]_补=[-1]_补=11111111$，$[N_2]_补=[+2]_补=00000010$，故$[N_1]_补-[N_2]_补=11111101$，所以无借位（$\overline{C}=0$），进位标志C＝1。按照上面的判断方法：C＝1，则$N_1 \geqslant N_2$，即$-1 \geqslant +2$，这显然是错误的。在比较带符号数大小时，应由符号位N和溢出位V两个标志位共同判断，判断方法是：当$[N_1]_补-[N_2]_补$后，N和V两个标志位相同时（即N＝1，V＝1或N＝0，V＝0），则$N_1 \geqslant N_2$；当N和V不同时（即N＝1，V＝0或N＝0，V＝1），则$N_1<N_2$。

另外，由于6502 CPU的比较指令CMP只影响标志位C，N，Z，不影响溢出标志V，所以，在进行两个带符号数大小的比较时不能用CMP指令，只能用SBC指令。

例8. 若例7的数组中是有符号数，试将其由小到大排序。

我们仍采用例7的排序方法，但判断两数大小时要根据

标志位N和V共同判断，不能用C判断。源程序如下：

```
        ORG   $300
        LDY   $1000
        STY   $06
        DEC   $06
LOOP1   LDX   $06
LOOP2   SEC              ；置进位位为1，相当于清借位
        LDA   $1000,Y
        PHA
        SBC   $1000,X
        BVC   SIGN       ；V=0,转SIGN,查N位
        BMI   NEXT       ；V=1，N=1，不交换，转
                           NEXT
EXCH    LDA   $1000,X    ；V=1.N=0,交换
        STA   $1000,Y
        PLA
        STA   $1000,X
        JMP   NEXT
SIGN    BMI   EXCH       ；V=0,N=1,交换，转EXCH
NEXT    DEX
        BNE   LOOP2
        LDY   $06
        DEC   $06
        BNE   LOOP1
        BRK
```

例9．数据块传送。

将存储在＄6020～＄6083单元中的第一个数据块传送到＄7020～＄7083单元中；将存储在＄6120～＄6183单元中的第二个数据块传送到＄7120～＄7183单元中。源程序如下：

```
          ORG  $1A
          DFB  $20, $60, $20, $61   ；存两个数据块的源首址
          ORG  $FA
          DFB  $20, $70, $20, $71   ；存目标地址
          ORG  $300
          LDX  #0                   ；置X初值为0
LOOP1     LDY  #$64                 ；数据块长度送Y
LOOP2     LDA  ($1A,X)              ；从源数据块取一个数
                                      据
          STA  ($FA,X)              ；传送到目标地址
          INC  $1A,X                ；修改源地址
          INC  $FA,X                ；修改目标地址
          DEY                       ；Y－1→Y
          BNE  LOOP2                ；Y≠0，未送完，返回
                                      继续
          INX                       ；Y＝0，第一数据块已
                                      送完，准备送第二块
          INX
          CPX  #4                   ；第二块送完了吗？
          BNE  LOOP1                ；未送完，返回继续
          BRK                       ；已送完，结束
```

在这个程序中，先用伪指令把两个数据块的源首址和目标地址分别存放在零页的空单元内，然后在LDA和STA指令中使用了先变址（X）间接寻址方式。这种方法在同时处理多个数据块的传送时十分简便。

例10．延时子程序。

因为执行每一条指令都需要时间，利用这个道理可编制出延时程序。下面是一个实用的延时子程序，累加器A中的参数按需要在主程序中给出。

```
WAIT    SEC              ；执行时间  2个周期。
WAIT1   PHA              ；执行时间  3个周期。
WAIT2   SBC   #01        ；执行时间  2个周期。
        BNE   WAIT2      ；执行时间  3个周期，不跳转时为2。
        PLA              ；执行时间  4个周期。
        SBC   #01        ；执行时间  2个周期。
        BNE   WAIT1      ；执行时间  3个周期，不跳转时为2。
        RTS              ；执行时间  6个周期。
```

该程序的延迟时间为：

$$\left\{\left[(2+3)\times\frac{1+A}{2}\times A-A+(3+4+2+3)\times A\right]\right.$$

$$\left.-1+2+6\right\}T=\frac{1}{2}(5A^2+27A+14)T$$

式中T是CEC-Ⅰ机的时钟周期。

例11．代码转换程序。

代码转换是微型计算机应用中经常遇到的问题。因为外设常以ASCII码、BCD码或各种专用代码的形式向计算机提供数据；计算机接收到这些代码后必须将它们转换成可供处理的代码形式；计算机处理完毕后，还必须将结果再转换成适当的代码形式输送给外设。所以，学习代码转换程序的设计十分重要。

问题：编一子程序，把累加器A中的一位16进制数的ASCII码转换成二进制数。

CEC-Ⅰ机中，一位16进制数，ASCII码，8421 BCD码和二进制数之间的关系如表4.1所示。

由表4.1可见，当A中16进制数的ASCII码≤＄B9时，它对应的二进制数为A－＄B0；当A中16进制数的ASCII

表4.1 16进制数、ASCII码、BCD码、二进制数之间的对应关系

十六进制数	ASCII码	8421 BCD 码	二进制数
$00	$B0	0000 0000	0000 0000
$01	$B1	0000 0001	0000 0001
$02	$B2	0000 0010	0000 0010
$03	$B3	0000 0011	0000 0011
$04	$B4	0000 0100	0000 0100
$05	$B5	0000 0101	0000 0101
$06	$B6	0000 0110	0000 0110
$07	$B7	0000 0111	0000 0111
$08	$B8	0000 1000	0000 1000
$09	$B9	0000 1001	0000 1001
$0A	$C1	0001 0000	0000 1010
$0B	$C2	0001 0001	0000 1011
$0C	$C3	0001 0010	0000 1100
$0D	$C4	0001 0011	0000 1101
$0E	$C5	0001 0100	0000 1110
$0F	$C6	0001 0101	0000 1111

码≥$C1时，它对应的二进制数为A－$B7。由此分析可设计出源程序如下：

```
        ORG   $300
VABIN   CMP   #$C1     ; A-$C1
        BCS   NEXT     ; 若A≥$C1, 转NEXT
```

```
         SEC               ；若A＜$C1，置进位位为1
         SBC   #$B0        ；A-$B0-C̄→A
         RTS               ；返回
NEXT     SEC               ；置进位位为1
         SBC   #$B7        ；A-$B7-C̄→A
         RTS               ；返回
```

例12．编一子程序，将累加器A中的一位16进制数转换成ASCII码。

由表4.1可见，当A中的16进制数X＜$0A时，X的ASCII码为A中内容加$B0；当X≥$0A时，X的ASCII码为A中内容加$B7，源程序如下：

```
         ORG   $300
UBASC    CMP   #$0A        ；A-$0A
         BCS   NEXT        ；A≥$0A，转NEXT
         CLC
         ADC   #$B0
         RTS
NEXT     CLC
         ADC   #$B7
         RTS
```

例13．将存于$1000单元的无符号二进制数转换成三位BCD码，转换后存于$1001和$1002单元（高位数存高地址，低位数存低地址），源程序如下：

```
         ORG   $300
         LDX   $1000          ；取二进制数→X
         STX   $06            ；转存于零页$06单元
         LDA   #00            ；A清0
         STA   $1002          ；存结果高位的单元清0
         CMP   $06            ；A-($06)
```

```
        BEQ   NEXT     ; 是零，转结束
LOOP1   SED            ; 非0继续，置标志D=1，表明以下
                         运算均为十进制运算
        CLC            ; 清进位,C=0
        ADC   #01      ; A+01+C→A
        BCC   NEXT1    ; 无进位转NEXT1
        INC   $1002    ; 有进位，高位加1
NEXT1   DEX            ; X-1→X
        BNE   LOOP1    ; X≠0继续
NEXT    STA   $1001    ; X=0，存结果低位
        RTS            ; 返回
```

此程序简单，但运算速度慢。若需转换的二进制数的位数为n，则要进行2^n-1次运算。

例14．将n位二进制数转换为M位十进制数的BCD码。

将二进制数转换为十进制数的基本方法是按权展开相加。设n位二进制数为$a_{n-1}a_{n-2}\cdots a_1a_0$，$m$位十进制数为$N$，则有：

$$N=\sum_{i=0}^{n-1}a_i2^i$$

$$=a_{n-1}2^{n-1}+a_{n-2}2^{n-2}+\cdots+a_12^1+a_02^0$$

$$=\underbrace{2(2(\cdots(2(}_{2^{n-1}个}2a_{n-1}+a_{n-2})+a_{n-3})+\cdots)+a_1)+a_0$$

$$=2(2(\cdots(2(2\cdot0+a_{n-1})+a_{n-2})+a_{n-3})+\cdots)+a_1)+a_0$$

由此式分析可知，我们可以设置一个累加器,其初始值为0，然后执行2乘累加器内容再加二进制数高位的操作（十进制相加)，这样逐次运算下去就可累加出N来。

另外，在转换前还应计算出n位二进制数最多能转换出

多少位十进制数，即m与n的关系。因为n位二进制数的最大数为2^n-1，m位十进制数的最大数为10^m-1，故有：

$2^n-1=10^m-1$，由此得$2^n=10^m$，所以$n\lg2=m\lg10$故$m=n\cdot\lg2\doteq0.3n$。

若$n=8$，则$m=2.4$位BCD码，所以要用两个字节存放十进制数，因为两位BCD码就要占用一个字节。

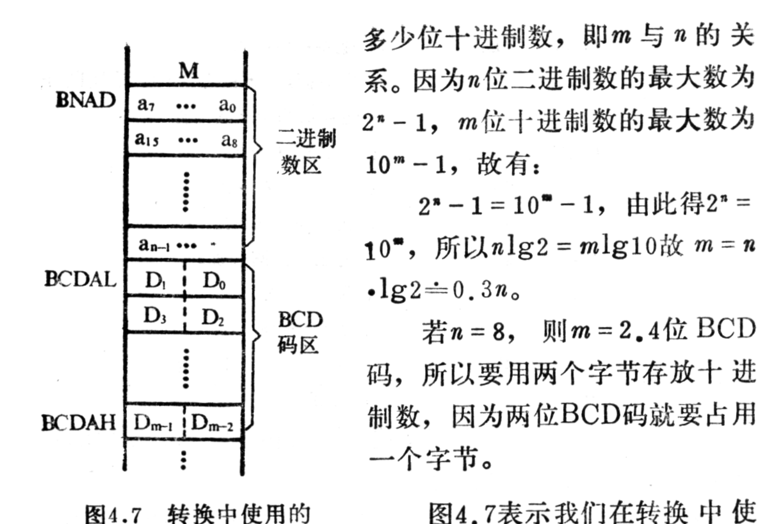


图4.7　转换中使用的存储区

图4.7表示我们在转换中使用的存储区。设二进制数的字节数为N，BCD码的字节数为M，源程序如下：

```
                                  ORG   $300
                                  LDY   #n        ；二进制位数
                                                    送Y
（清BCD码区）                      LDA   #00
                                  TAX
                        LOOP1     STA   BCDAL,X
                                  INX
                                  CPX   #M        ；M是BCD码
                                                    区字节数
                                  BNE   LOOP1
（整个二进制数左移一位）LOOP2     LDX   #00
                                  CLC
                                  ASL   BNAD,X
```

```
                 INX
                 CPX   #N        ；N为二进制字
                                   节数
                 BNE   LOOP2     ；未移完继续，
                                   移完往下执
                                   行，最高位在
                                   C中
（2*BCD码+进位）   LDX   #00
                 SED
          LOOP3  LDA   BCDAL,X
                 ADC   BCDAL,X
                 INX
                 CPX   #M
                 BNE   LOOP3
                 DEY
                 BNE   LOOP2
                 RTS
```

例15. 电子琴程序。

CEC-Ⅰ机中有一个小喇叭，运行以下程序后，可以定义键盘上的A、B、C、D、E、F、G键分别与低八度的音调相对应，^A～^G分别与中音的A～G音调相对应。这样，当按一个键时，喇叭里将发出相应的声音，可以用来演奏歌曲。在CEC-Ⅰ机的监控程序中有一个延时子程序WAIT，它的延迟时间：

$$T=\frac{1}{2}(26+27A+5A^2)*1.023\mu s$$

式中A是累加器的值。表4.2列出了采用WAIT子程序发出钢琴的中音C调到低音C调一共十四个音调的频率以及所要

表4.2 音调与A值的关系

中音	频率(Hz)	A值(十六进制)	低音	频率(Hz)	A值(十六进制)
C	261.7	13	C	130.8	26
D	293.7	12	D	146.8	24
E	329.6	11	E	164.8	22
F	349.2	10	F	174.6	20
G	392.0	0F	G	196.0	1E
A	440.0	0E	A	220.0	1C
B	493.9	0D	B	246.9	1A

求的A值。

从CEC-Ⅰ的ASCII码表可知，A～G对应的ASCII码为＄C1～＄C7；^A～^G对应的ASCII码为＄81～＄87。

电子琴程序的流程图如图4.8所示。

源程序如下：

```
KEYIN    EQU   $FD1B
WAIT     EQU   $FCA8
         ORG   $400
MUSIC    JSR   KEYIN        ；存闭合键的ASCII码在A中
         CMP   #$C1         ；是低音吗?
         BCC   MIXCT        ；小于$C1跳转到中音
         CMP   #$C8         ；大于$C7是按错了键，重来
         BCS   MUSIC        ；
```

```
        AND   #$07              ;计算变址值
        JMP   NEXT
MIXCT   CMP   #$81              ;小于$81是按错键，重来
        BCC   MUSIC
        CMP   #$88              ;大于$87，小于$C1时，重来
        BCS   MUSIC
        AND   #$07
        ADC   #$07              ;计算变址值
NEXT    TAX
        LDA   LABL-1,X          ;从表4.2中取出按键对应的A值
        TAY
LOOP    JSR   WAIT              ;转延时程序
        STA   $C030             ;访问触发器，使其翻转，驱动喇
                                 叭发声
        BIT   $C000             ;测试有新的键被按吗?
        BMI   MUSIC             ;若有，程序重新开始
        TYA                     ;没有，则继续发原来的音
        JMP   LOOP
LABL    DFB   $0E, $0D, $13, $12, $11, $10, $0F
        DFB   $1C, $1A, $26, $24, $22, $20, $1E
```

此数据表中的数据是根据表4.2中A～G和^A～^G的顺序存放的。$C030是驱动喇叭发音的触发器的入口地址；$C000是键盘输入单元，当有键闭合时，该单元的Bit7为1，所以用BMI指令来考查是否有新的键被按下。KEYIN是监控程序中的一个子程序，它的作用是读键盘，将闭合键的ASCII码存入A中。表4.2中的频率和A值还可根据你自己选定的音调和延时程序重新计算。

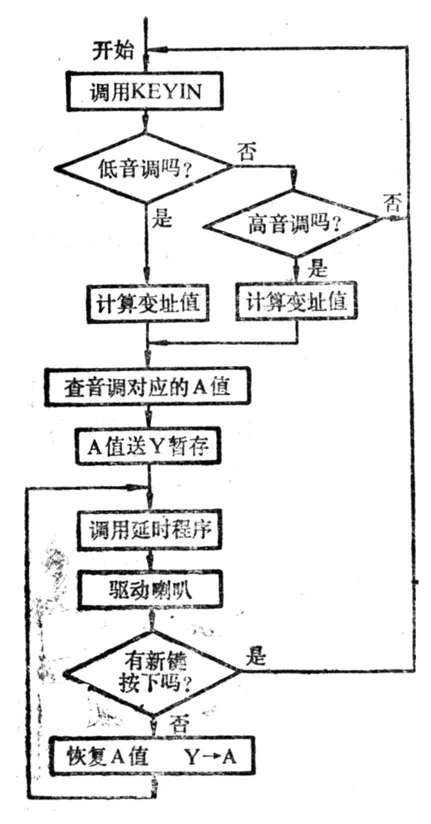


图4.8　电子琴程序框图

习　题

1. 利用变址寄存器，把自＄1000单元开始的100个数传送到自＄1080单元开始的存储区。请编一程序实现上述

目的。

2．自＄1000单元开始存有100个数，要求编一程序，把它传送到＄1050开始的区域（注意：数据有重叠区）。

3．编一程序，把题1中的数据块中的负数传送到自＄1070单元开始的存储区中，并计算负数的个数，将负数个数存入＄106F单元。

4．有一串无符号数存放在从＄1103单元开始的存储区中，数的个数存放在＄1102单元中。编一程序，求这串无符号数的和。这个和可能是16位无符号数，请将和存入＄1100和＄1101单元。

5．编一程序，找出一个数据块中的最大无符号数。设数据块中的元素个数存于＄1001单元中，数据块从＄1002单元开始存放，找出的最大数放入＄1000单元中。

6．若题5中是带符号数，编一程序找出它的最大数，存入＄1000单元中。

7．有两个四字节二进制数 N_1 和 N_2，分别从＄1000和＄1004单元开始存放。编一程序计算 N_1-N_2 之差，并把结果存入从＄1000开始的单元中。

8．将题7中的二进制数改为十进制数，求它们的差，并把结果存入＄1000开始的单元中。

9．请利用CEC-Ⅰ监控系统中的WAIT子程序，编制一个延时1秒钟的延时程序。

10．编一程序，将一个单元的BCD码转换为它的ASCII码，存入你设定的两个单元中。

11．若有从键盘输入的0～9之间的10个字符的ASCII码，存放在以BUFFER为起始地址的相邻十个单元中，

编一程序把它们转换为BCD码，存放在从BUFFER开始的五个相邻单元中。

12．若在＄1050单元中有一个数X，请把此数高四位变0，低四位不变，并送回原单元。

13．把题12中的数的高四位置1，低四位不变，送回原单元。

14．把题12中的数的高四位取反，低四位不变，送回原单元。

15．若从＄1000单元开始存了100个数。编一程序检查这些数，正数保持不变，负数取补后送回原单元。

16．在＄1000单元中存有一个无符号数X，请用移位和相加的方法实现X＊10后送回原单元（假设乘积≤255）。

17．假设从＄1000单元开始存有一个字符串，以字符“空格”为开始，以字符“Ω”为结束。编一程序统计字符串长度（不计前导“空格”和结尾“Ω”字符），并将长度存＄06单元（假设长度≤255）。

18．统计题17中数字字符（“0”～“9”）的个数，并将统计结果存＄07单元（假设个数≤255）。

第五章　源程序的编辑和运行

我们在前面已经讲过，用汇编语言编写的源程序必须翻译成机器语言程序（即目标程序），才能在计算机中运行。完成这种翻译工作的软件称为汇编程序，这种翻译工作称为汇编。汇编时，首先必须用编辑软件（EDITOR）将源程序送入计算机，形成源程序的ASCII码文件（常称为源文件），存放在磁盘上，这个过程称为编辑。然后调用汇编程序对磁盘上的源文件进行汇编，汇编后形成一个目标文件（目标程序）送入磁盘，然后才能运行该程序。CEC-Ⅰ机使用的是6502微处理器，与APPIE-Ⅱ机具有相同CPU，在它们中编辑程序和汇编程序已经合并为一个文件，称为EDASM（编辑汇编）文件。下面，我们将介绍在CEC-Ⅰ机中如何使用这个EDASM文件。

在以下讨论中需要用到CEC-Ⅰ机中的下述提示符：

］　BASIC状态提示符。

*　监控状态提示符。

!　小汇编状态提示符。

:　编辑汇编状态提示符。

5.1　编辑汇编程序的启动

CEC-Ⅰ中华学习机可以使用APPIE-Ⅱ的EDASM软

件进行编辑汇编操作。这个文件使用比较广泛，容易找到，现介绍如下。

启动方法是：

在软盘驱动器内插入APPIE-Ⅱ EDASM盘片，然后开启电源，则立刻可听见喇叭声响，键盘右上角指示灯亮，磁盘驱动器指示灯亮。待磁盘驱动器指示灯灭后，屏幕上出现以下信息：

APPLE Ⅱ EDITOR-ASSEMBLER

CURRENT ASSEMBLER ID STAMP IS :

|5|-JUN-83 #0000414

此时，光标在|5|字上闪烁，敲回车键，待屏幕显示出：

:_

屏幕左下角出现“:”时表示编辑汇编程序安装成功，系统进入EDASM状态。“_”为闪烁的光标，系统等待用户输入命令。

5.2 如何编辑源程序

要在计算机中运行您编好的汇编语言源程序，必须首先用EDASM软件的编辑命令将源程序送入计算机，并进行必要的编辑和修改，然后再对源程序进行汇编，汇编成目标程序后再运行。本节主要介绍EDASM软件中的编辑命令及使用方法。

1. 键盘输入源程序命令

A——使用此命令允许用户通过键盘向计算机输入汇编源程序。

例1. 将＄1000和＄1001两个存储单元的内容进行十进制相加后，存入＄1002单元中。源程序输入过程如下：

```
:A↙
   1 ␣ORG␣ $0300 ↙
   2 START␣SED ↙
   3 ␣CLC ↙
   4 ␣LDA␣ $1000 ↙
   5 ␣ADC␣ $1001 ↙
   6 ␣STA␣ $1002 ↙
   7 ␣BRK ↙
   8 [CTRL-D] ↙
:                    ；结束输入，返回:状态
```

在输入源程序时必须注意以下几点：

① 每行的行号是机器自动显示的，有下划线的部分由操作者键入。

② "↙"表示敲回车键；"␣"表示敲空格键。

③ [CTRL-D]表示先按住[CTRL]键，再敲D键。以下将它们简单表示为^D，也可用^Q，它们的作用都是结束源程序的输入，回到:状态。

④ 在各行键入汇编语句时，若该语句无标号，则应在行号后敲一空格键后，再键入语句（此空格经编辑后将自动留出八个字符位置）；若有标号，则紧跟行号后打入标号，行号和标号间不敲空格，如上例中第2行。

⑤ 每个程序输入完毕后均应敲^D或^Q键，以便退出A命令，回到:状态。

2、列表显示源程序命令（LIST）

L 行号——此命令用于显示已送入内存的文本。

例2．显示例1中源程序的1～5行。

```
: L ␣ 1-5 ↙ (以下是屏幕显示的结果)
  1          ORG  $0300
  2   START  SED
  3          CLC
  4          LDA  $1000
  5          ADC  $1001
:
```

例3．显示文本所有行。

```
: L ↙
  1          ORG  $0300
  2   START  SED
  3          CLC
  4          LDA  $1000
  5          ADC  $1001
  6          STA  $1002
  7          BRK
:
```

3．打印（PRINT）源程序命令

P 行号——作用同L命令，区别仅在于P命令不显示行号。如：

```
: P 1—3 ↙
             ORG  $0300
      START  SED
             CLC
```

:

4. 修改（CHANGE）命令

C——修改文本的某行。格式为：

:C行号·旧字符串·新字符串↵

或 :C行号$旧字符串$新字符串↵

此命令的含义是用新字符串代替旧字符串。按上述格式键入C命令后，系统要求回答以下询问：

ALL OR SOME? (A/S)? （全部或部分）

若敲A键回答，则表示该行中C命令指出的所有旧字符串都要用新字符串代替；若键入S回答，则表示只修改部分旧字符串。对要修改的旧字符串，需要再敲一次S键，系统才能用新字符串替换该旧字符串；对于需要保留的旧字符串，则敲 ESC 键跳过。

例4. 将例1第1行中的字符0改为字符6。

```
: L 1 ↙                      ；显示第1行，以便下面观察修
                                改情况
        1     ORG   $0300
: C1.0.6 ↙                   ；将第1行中的0改为6
  ALL OR SOME? (A/S)?        ；全改还是只改部分？
_                            ；光标闪烁位置，等待回答A
                                或S
```

此时若敲A键作为回答，则第1行中所有字符0都改为6，并显示出修改结果为：

```
        1     ORG   $6366
: _
```

若敲S键回答，则表示只修改部分“0”字符。此时，若再按一次S键，则修改第一个0为6；再按一次S键，则修

改第二个0为6；再按 ESC 键，则第三个0不修改。此时可用L命令显示修改结果：

```
: L 1 ↙
          1    ORG   $6360  ；只修改以前两个0
: _
```

若在按S键回答后不再继续按S键，只按 ESC 键，则不作任何修改。此时用L命令得：

```
: L 1 ↙
          1    ORG   $0300  ；全部未改
: _
```

5．删除（DELETE）命令

D 行号——用于删除文本中某一行或某连续段。如：

```
: D1↙          ；删除第1行
: D1—5↙        ；删除第1—5行
```

6．插入（INSERT）命令

I 行号——用于插入一行语句。

例5．在例1的第三行插入NOP。操作如下：

```
: I 3 ↙                 ；机器立即空出第三行，等待插入
       3 ␣ NOP ↙        ；在第三行插入NOP
       4 _              ；还可继续在第四行插入
```

如果不需要继续插入，则按^D，退出I命令。此时可用L命令检查插入后的情况：

```
: L ↙
       1             ORG   $0300
       2  START      SED
       3             NOP          ；第三行已插入NOP
```

```
4          CLC        ；其余行号已重新排列
5          LDA  $1000
6          ADC  $1001
7          STA  $1002
8          BRK
:_
```

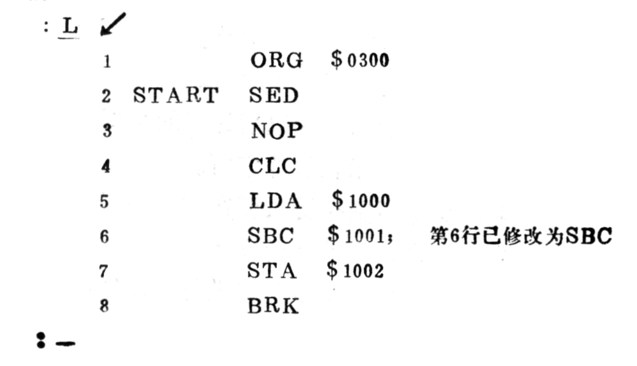
7．替换（REPLACE）命令

R　行号——对文本中某行进行重写。

例6．将上例中的ADC指令改为SBC指令。

```
:R 6 ↙                          ；修改第6行。
     6 ␣ SBC $1001 ↙
     7 _                        ；还可继续修改第7行。
```

如果不需要继续修改，则按^D退出R命令。下面用L命令检查修改情况：



```
:L ↙
   1          ORG   $0300
   2  START   SED
   3          NOP
   4          CLC
   5          LDA   $1000
   6          SBC   $1001；   第6行已修改为SBC
   7          STA   $1002
   8          BRK
:_
```

8．查找（FIND）字符命令

F——查找文本中某行或某连续段。格式为：

① F　行号；查找某行

② F・字符串・　或

F$字符串$　；查找命令中指定字符串的所在行。

例7.

```
:F 1 ↙                   ；查找上例中第一行
 1    ORG   $0300        ；查到后显示之
:F·STA· ↙                ；查找上例中含字符串STA的所
                           有行
 2  START SED            ；第二行中含STA
 7           STA  $1002  ；第七行中含STA
```

9．编辑（EDIT）命令

E　行号——对源程序中的某行进行修改。

在E命令中可以用以下光标控制键：

→　　光标右移一个字符。

←　　光标左移一个字符。

↵　　编辑结束，编辑有效，退出E命令，回到EDA-SM。

^D　删去光标指定位置的字符。

^I　在光标指定位置插入字符。

^T　删去光标指示位置以后该行的全部字符。

^R　改写光标指示位置的字符。

^F　在此命令后紧接键入要在该行中寻找的字 符，则光标就自动移动到所寻字符位置上。

^X　退出E命令，废除正在进行的操作，回到EDA-SM状态。

利用E命令和上述光标控制键，可以方便地对源程序行进行修改。

10．存盘命令（SAVE）

SAVE　文件名——将源程序用指定的名字保 存 在 磁盘上。

当用编辑命令编辑好一个用户的源文件以后，就可以把这个源文件存入磁盘。存盘时要给源文件取一个名字。比如，我们把上面的练习程序取名为SHUXI，然后用SAVE命令存盘，即：

: SAVE SHUXI ↙

这样，名字为SHUXI.EDASM的文件就存入磁盘了，此时可用CATALOG命令列出磁盘目录，检查文件是否已经存入盘中。

: CATALOG ↙

这时屏幕上显示的磁盘目录中已有一个名字为SHUXI.EDASM的文件存在了。

11. 文件载入命令（LOAD）

LOAD 文件名——将命令中指定的文件从磁盘调入内存。如：

: LOAD SHUXI ↙ ；执行该命令就将名字为SHUXI的文件调入内存中。

5.3 源程序的汇编

1. 如何汇编源程序

经过编辑并存入磁盘的源程序，用以下汇编命令进行汇编。汇编命令格式为：

ASM 源文件名

该汇编命令的作用是对磁盘上指定的文件进行汇编，将源程序翻译成目标程序。

例8. 将SHUXI进行汇编。其操作过程如下：

: ASM SHUXI ↙

此时屏幕提示：

PRESS ANY KEY TO CONTINUE _ (敲任何键继续)

此时敲任一键作为回答，机器就开始对盘上的SHUXI文件进行汇编，并在屏幕上显示出源文件名SHUXI、汇编后的目标文件名SHUXI.OBJ以及汇编后的程序清单。显示情况如下：

```
SOURCE FILE: SHUXI
---- NEXT OBJECT FILE NAME IS SHUXI. OBJ0
0300:                1         ORG   $0300
0300:F8              2 START   SED
0301:18              3         CLC
0302:AD 00 10        4         LDA   $1000
0305:6D 01 10        5         ADC   $1001
0308:8D 02 10        6         STA   $1002
030B:00              7         BRK
*** SUCCESSFUL ASSEMBLY: NO ERRORS
?0300 START                   ?0300 START
```

汇编结束后敲任一键，系统就退出汇编，回到EDASM状态，屏幕上出现提示符：

: _

由上面的程序清单看到：清单左边第一列是汇编后的目标程序存放的地址。第二列是源程序中每条符号指令的机器码，也就是目标程序。比如SED指令的机器码是F8（16进制表示，下同），汇编后存于0300（16进制表示，下同）单元；LDA $1000指令的机器码是AD 00 10，它们分别依次存于0302，0303，0304单元。ORG $0300是伪指令，因此没有相应的机器码，它的作用是指示汇编程序将它下面

的源程序汇编后，把目标程序从＄0300单元开始存放，正如清单第一列和第二列显示出来的那样。第三列是源程序的行号，第四列是标号，第五列是源程序。清单下面第一行是指出“汇编成功，没有错误”；再下面一行是指出标号START经汇编后对应的绝对地址为0300单元。

清单中出现的英文提示的意思如下：

SOURCE FILE 源文件

NEXT OBJECT FILE NAME IS SHUXI.OBJ0 下面的目标文件名字是 SHUXI. OBJ0

SUCCESSFUL ASSEMBLY 汇编成功

NO ERRORS 没有错误

2. 如何打印汇编清单

如果希望在汇编过程中打印出清单，可以按如下步骤操作：

① 打开打印机电源。

② 键入以下命令

：PR#1，L××P×× ↙

：ASM 源文件名 ↙

这样，在汇编过程中就可打印出汇编的清单。前一个命令中的L××表示一页中要打印的行数，P××表示一页中的总行数。比如L16P20，表示这一页共20行，打印16行之后空四行。如果L××中填的数大于P××中填的数，则打印时就不分页了，而是连续打印下去。

5.4 目标程序的运行

仍以SHUXI为例，若要运行它的目标文件SHUXI.

OBJ0，其操作步骤如下：

```
: END ↙                  ；执行此命令，使系统进入BASIC状态
] BRUN SHUXI.OBJ0 ↙      ；在BASIC状态下运行目标程序
 030E- A=99 X=98 Y=25 P=BC S=EE
*_                       ；运行完毕，显示CPU各寄存器的内容，
                           进入监控状态
```

若要在监控状态下运行此程序时，用以下命令（此时因目标程序已在内存中，不用再调盘）：

```
*0300G ↙
```

这里0300是SHUXI.OBJ0的首地址，在这条命令中可以省去十六进制标志＄。

有时会遇到源程序被汇编后产生的目标文件不止一个，如第四章中例9，由于程序用ORG指令分成了三段，因此将它汇编后产生的目标文件是三个。假如源文件名是NUMBER，则三个目标文件分别为NUMBER.OBJ0，NUMBER.OBJ1，NUMBER.OBJ2。运行它时，要首先将这三个目标文件从磁盘调入内存后才能运行。调入内存用以下命令：

```
] BLOAD NUMBER. OBJ0 ↙
] BLOAD NUMBER. OBJ1 ↙
] BLOAD NUMBER. OBJ2 ↙
] _
```

这样，就把三个目标文件都装入了内存，然后键入以下命令：

```
] CALL-151 ↙     ；使系统进入监控状态
*300G ↙          ；从目标程序的首地址开始运行
*_
```

文件运行完毕后可用监控命令检查运行结果（监控命令的使

用参见下一章）。

5.5 CEC-I的小汇编

当你的CEC-I机尚未配置软盘驱动器，或者虽然配置了软盘驱动器却又没有EDASM编辑汇编软件时，可以使用已经固化在CEC-I机中的小汇编程序。这是一种简单的汇编工具软件，可以将输入的汇编源程序的符号指令逐条翻译成机器指令，存放在指定的内存单元并显示出来，这就省去了你自己去查表翻译机器码的麻烦。小汇编程序不认识伪指令，所以使用小汇编时不能用标号和符号地址等，因而小汇编程序只能用来汇编一些简单的源程序。

5.5.1 如何进入和退出小汇编状态

1. 进入小汇编

① 在监控状态下进入小汇编的操作

```
*D350G ↙
!                    ;! 是已进入小汇编的状态标志。
```

② 在BASIC状态下进入小汇编的操作

```
]CALL-151 ↙
*D350G ↙
!
```

2. 如何退出小汇编状态

① 退出小汇编返回监控程序

```
! $D360G ↙
*
```

或

```
! $FF69G ↙
```

*　　　　　　　；返回监控程序。

② 退出小汇编返回BASIC状态

! $35D0G ↙

]

5.5.2 如何使用小汇编

1. 小汇编命令的格式和用法

地址：符号指令——其功能是将命令中输入的一条符号指令翻译成机器码，并存放在命令指定的地址中；若是多字节指令，则将其机器码存放在以该地址为首地址的连续几个存储单元中。

例9.

! 300:CLC ↙　　　　；输入一条符号指令，汇编一条指令

0300- 18　CLC　　　；这是显示的汇编结果。CLC的机器码是18(十六进制表示，下同)，已存放在命令指定的0300(十六进制表示，下同)单元中

! 300:LDA #$FF ↙

0300- A9 FF LDA #$FF；LDA #$FF的机器码是A9，FF，存放在0300开始的连续两个单元中，即A9存入0300单元，FF存入0301单元。

如果下一条指令要接着上一条指令存放，则可不再输入地址，而在输入"地址"位置打入一个空格，紧接着输入符号指令即可，此时小汇编会自动计算下一个地址。这样，只要设定了起始地址之后，就可以一行一行地连续输入一系列的汇编语句。

例10. 输入一个完整的汇编源程序。

! 300:SED ↙

0300- F8　　SED；输入一条，汇编一条

! ␣ CLC ↙

0301- 18　　CLC；空格␣表示接续上面的地址存放机器码

```
! ␣ LDA # $88 ↙
  0302- A9 88 LDA #$88
! ␣ ADC #$04 ↙
  0304- 69 04   ADC #$04
! ␣ BRK ↙

  0306- 00    BRK
!
```

输入完毕就汇编完毕。此时可用命令显示：

```
! $300L ↙          ；这里，命令中的$不能省去
```

执行这条命令，小汇编程序就把自$0300单元开始存放的目标程序进行反汇编，并显示出来。一次显示20行，不足20行用00填补。下面是执行$300L命令后屏幕上显示的结果。

```
! $300L ↙
  0300- F8        SED                        （第一行）
  0301- 18        CLC
  0302- A9 88     LDA     #$88
  0304- 69 04     ADC     #$04
  0306- 00        BRK
  0307- 00        BRK
     ⋮             ⋮
  0313- 00        BRK                        （第20行）
```

2．在小汇编状态下使用监控命令

在小汇编状态下可以使用CEC-I机的各种监控命令，这是十分方便的。使用时只要在各监控命令前加上符号“$”，就变成了小汇编状态下的命令。

比如，在小汇编状态下连续运行目标程序的命令格式为：

! $首地址G ↙

命令中的首地址是指要运行的目标程序的首地址。

例11．运行例10的目标程序。

! $300G ↙

这样，就连续运行从$300单元开始存放的目标程序，直到BRK中断为止，并显示出运行结束时各寄存器的状态。显示如下：

0308- A= X= Y= P= S=

例12．修改某单元内容。

! $300:D8 ↙

这条命令的意思是将$0300单元的内容改为D8（十六进制表示）。D8是CLD指令的机器码，实际上就是把例10中的SED指令改成了CLD指令。

CEC-I机有许多监控命令，详见第六章。在每一条监控命令前加上“$”，就变成了在小汇编状态下能执行的命令。

3．使用小汇编的注意事项

① 小汇编不认识伪指令。所以使用小汇编时不能用标号和符号地址，只能给出具体数据和绝对地址。

② 转移指令中的地址码部分，只能用目标地址的绝对地址码，不能用符号地址代替，也不能用转移步长。如：

! 310 : JSR $C100 ↙

这里的$C100是要转移去的目标地址的绝对地址码，它不能用标号代替，也不能用转移步长代替。

③ 符号指令中的操作数或操作数地址只能用16进制数不能用其它任一进制数，因为小汇编程序只认识十六进制数，不认识其它进制数。

习　题

1. 什么叫汇编语言源程序？什么叫汇编程序？

2. 为什么汇编源程序一定要汇编成目标程序后计算机才能执行？可以用哪些方法将源程序汇编成目标程序？

3. 什么叫伪指令？请指出经过汇编后下述各标号的数值是多少？JSR指令执行后PC＝？JMP指令执行后PC＝？

```
               ORG   $6000
WORK1          DFB   $1C, $8A, $23, $5D, $6E
WORK2          DW    $C080
WORK3          DFB   'A, 'B, 'C, 'D
BUFFER         DS    15
COUNT          EQU   $808E
               JSR   COUNT
               JMP   (WORK2)
```

4. 试在CEC-I机上用EDASM软件将下述程序汇编成目标程序，并打印出汇编清单；再运行目标程序，打印出运行结果。源程序如下：

```
ORG   $06
DFB   $35, $FA, $00
ORG   $300
CLC
CLD
LDX   #00
LDA   $06, X
INX
ADC   $06, X
STA   ($06,X)
BRK
```

5. 试用小汇编程序将题4中的程序汇编成目标程序，再运行目标程序，记录运行结果。使用小汇编时，伪指令ORG、DFB作何处理？

第六章 监控系统

监控系统是与计算机结构密切联系的一种管理计算机的系统程序，是人们使用计算机的重要工具。CEC-Ⅰ机的监控程序已被固化在ROM中。监控程序的功能体现在监控命令上，通过监控命令提供对系统的基本操作。比如：管理键盘输入，字符显示，检查、修改、比较寄存器和内存单元的内容，运行目标程序，跟踪程序执行轨迹，反汇编，对盒式录音机存取数据，等等。此外，监控系统中还有不少优化的子程序可供用户直接调用。

6.1 监控程序的结构

CEC-Ⅰ学习机的监控程序已固化在ROM中，与APPLE SOFT BASIC解释程序一起，合用一块27256 ROM芯片U_7，地址为$D000～$FFFF，并使用了RAM中若干存储单元。

当机器加电（冷启动）或在键盘上按下^-RESET键（热启动）时，首先进入系统初始化程序：清除各种标志和显示缓冲区，设置各种必要的工作方式，如果一切正常，主机将自动进入西文BASIC状态，屏幕显示如图6.1所示。图中]是BASIC状态提示符；▨为光标，指示您输入字符的显示位置，等待您按键输入。

ZHONG HUA XUE XI JI

VERSION 1.1

] ▨

图6.1 CEC-Ⅰ学习机的启动

如果此时键入：

]CALL-151 ↙

*

则系统立即进入监控状态，屏幕上显示出监控标志“*”。

图6.2给出了CEC-Ⅰ学习机系统初始化的流程，图6.3是CEC-Ⅰ学习机的监控命令处理程序的流程。

6.2 监控程序占用的RAM工作区

1. *在零页使用的系统工作单元*

监控程序在工作过程中需要使用RAM中的一些单元作为它的系统工作区。下面列出它在零页使用的工作单元及对应的监控程序符号。

$00	LOC0	$24	CH
$01	LOC1	$25	CV
$20	WNDLFT	$26	GBASL
$21	WNDWDTH	$27	GBASH
$22	WNDTOP	$28	BASL
$23	WNDBTM	$29	BASH

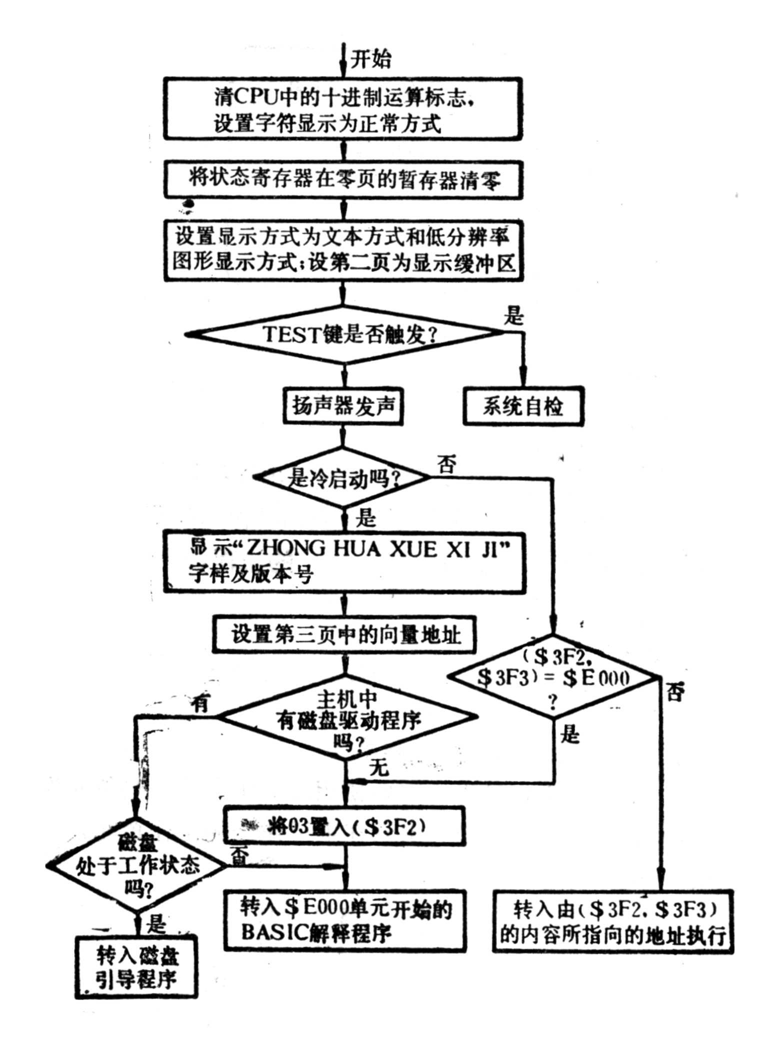


图6.2 CEC-I监控系统的初始化程序流程

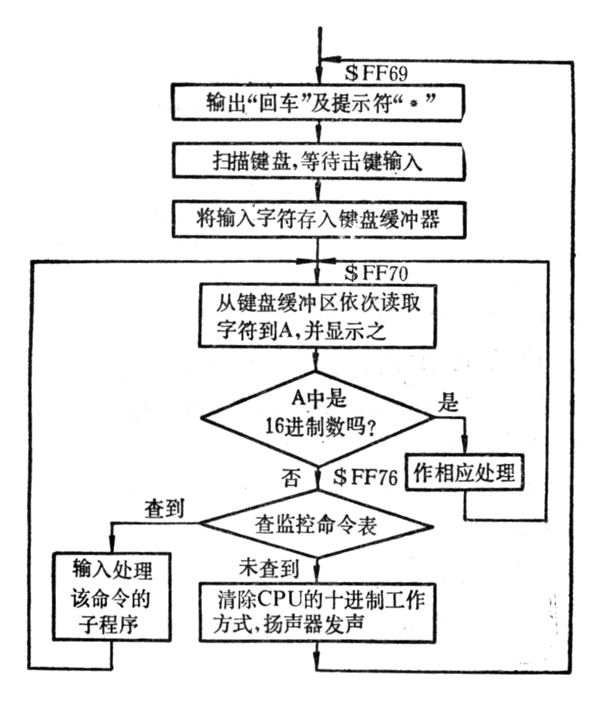


图6.3 监控命令处理程序流程图

$2A	BAS2L	$33	PROMPT
$2B	BAS2H	$34	YSAV
$2C	H2, LMNEM	$35	YSAV1
$2D	V2, RMNEM	$36	CSWL
$2E	MASK, FORMAT, CHKSUM	$37	CSWH
		$38	KSWL
$2F	LASTIN, LENGTH	$39	KSWH
$30	COLOR	$3A	PCL
$31	MODE	$3B	PCH
$32	INVFLG	$3C	A1L

$3D　A1H

$3E　A2L

$3F　A2H

$40　A3L

$41　A3H

$42　A4L

$43　A4H

$44　A5L

$45　A5H，ACC

$46　XREG

$47　YREG

$48　STATUS

$49　SPNT

$4E　RNDL

$4F　RNDH

现将上述单元中一些常用单元的作用介始如下：

$20（WNDLFT）：文本窗口左边框位置单元。CEC-Ⅰ机把屏幕从左到右分为40列，列号为0～39。第0列是屏幕最左边一列，第39列是屏幕最右边一列。若置$20单元的内容为5，则文本显示将从屏幕的第5列位置开始。开机后，监控程序置$20单元为0。

$21（WNDWDTH）：文本窗口宽度单元。单元内的值确定屏幕上所显示的文本宽度，宽度的值在1～40之间。开机后，监控程序置$21单元为40。

$22（WNDTOP）：文本窗口的上边框单元。单元内的值确定屏幕上显示的文本的上边框位置。CEC-Ⅰ机把屏幕分成24行（0～23），一般置$22单元的值为0。

$23（WNDBTM）：文本窗口的下边框单元，其值在0～23之间选择，它确定在屏幕上显示文本的下边框位置。

$24（CH）：光标的水平位置单元。单元的值确定了当前光标在屏幕上的水平位置。

$25（CV）：光标的垂直位置单元。单元的值确定了光标在屏幕上的垂直位置。

$28（BASL），$29（BASH）：存放光标所在行位置对

应的内存显示缓冲区基地址。

$ 2C(H2,LMNEM)：画低分辨率图形水平线的右边位置单元。

$ 2D(V2,RMNEM)：画低分辨率图形垂直线的下边位置单元。

$ 30 (COLOR)：低分辨率图形的色彩单元。单元的值确定了低分辨率图形的色彩。可选色彩共16种，它们与 $ 30 单元的值的对应关系如表6.1所示。

表6.1 颜色代码

数值（16进制）	颜　色	数值（16进制）	颜　色
0	黑	8	棕色
1	洋红	9	桔黄
2	深蓝	A	灰色2
3	紫色	B	粉红
4	深绿	C	浅绿
5	灰色1	D	黄色
6	蓝色	E	海蓝色
7	浅蓝	F	白色

$ 32 (INVFLG)：决定字符显示方式的单元。如果($ 32) = FF（十六进制，下同)，字符显示为正常方式，即背景为黑色，字符为白色；($ 32) = 3F，字符显示为反相方式，即背景为白色，字符为黑色；($ 32) = 7F，则字符显示为闪烁方式，即显示的字符不停地闪烁。

$ 36 (CSWL)，$ 37(CSWH)：输出子程序的转接口

单元。

$38 (KSWL), $39 (KSWH):输入子程序的转接口单元。即在这两个单元中存放输入子程序的入口地址。

$3C(A1L)～$45(A5H, ACC):监控命令中输入的地址存放单元。

2. 在第三页使用的系统工作单元

监控程序使用第三页的情况见表6.2。

表6.2 监控程序使用第三页的情况

地址	功能
$03F0 $03F1	BRK指令的中断向量地址，即存放BRK请求子程序入口地址的地址
$03F2 $03F3	复位的中断向量地址
$03F4	加电标志
$03F5 $03F6 $03F7	存放转向处理APPLE SOFT命令的转移指令的地址
$03F8 $03F9 $03FA	存放转向处理用户命令（^Y）的转移指令的地址
$03FB $03FC $03FD	存放转向处理非屏蔽中断服务程序的转移指令的地址
$03FE $03FF	可屏蔽中断请求的中断向量地址，即存放IRQ中断服务程序入口地址的地址

6.3 监 控 命 令

6.3.1 如何进入和退出监控状态

1. 在BASIC状态下进入监控状态

```
]CALL-151 ↙

*                    ；*是监控状态标志
```

2. 在监控状态下返回BASIC状态

```
* ^B ↙     或    （^表示ctrl，下同）

* ^C ↙

]                    ；已返回BASIC状态。
```

执行^B命令和^C命令都可使系统从监控状态返回BASIC状态，但^B将使以前使用的BASIC程序及变量全部消失；^C命令则能保存原来的BASIC程序及变量。

此外，还可执行以下操作返回BASIC状态。

```
* 3D0G ↙

]
```

6.3.2 监控命令的格式与用法

在键入监控命令时，若输入的十六进制地址少于四位，监控程序会在它的前面添0补足四位；若多于四位，则只取后四位。监控命令执行时显示的内容均用十六进制数表示。监控命令行的长度受系统键盘缓冲区的限制，不得多于254个字符，否则监控系统将从本行跳出，并忽略该行键入的数据和

命令。

1．检查存储器的内容

格式1：＊地址↲ 显示该地址单元的内容。

格式2：＊．地址2↲ 显示从当前地址开始到地址2的各存储单元内容。

格式3：＊地址1．地址2↲ 显示从地址1到地址2的各存储单元内容。

格式4：＊↲ 只敲回车键，显示从当前地址开始的八个单元内容。

例1．＊C00↲ 检查＄0C00单元的内容。

0C00- 11 这是显示情况。

例2．＊．C10↲ 检查从当前地址＄0C00单元开始到＄0C10单元的内容。

```
0C01-    00 13 45 C8 D0 F3 A7
0C08- 17 89 08 A0 0B 91 7A 8C
```

这是显示情况，每行显示八个单元内容。

例3．＊0C20．0C30↲ 检查从＄0C20至＄0C30单元的内容。

```
0C20- 01 23 51 08 9A 8B 7A 7B
0C28- 10 22 15 8A 9B 7C 8B 9F
0C30- FD
```

2．修改（或输入）内存单元的内容

格式1：＊地址：数据数␣数据␣数据␣…↲ 从地址指定单元开始，依次将所列数据送入存储器（修改原内容）。（␣表示一个空格，下同）

格式2：＊：数据␣数据␣数据␣…↲ 接续前面的地址

将数据继续送入内存。

例4. *C00:00 01 02 ED FC 08↵ 从$0C00单元开始连续修改6个单元的内容，用这六个新输入的数据代替原来的数据。此时可用以下命令检查修改后的情况：

* C00.C05 ↙

0C00- 00 01 02 ED FC 08 （说明修改正确）

例5. 接续前面的地址继续修改（或输入）

* : 03 04 05 06 07 08 0A ↙

* C06.C0C ↙

0C06- 03 04

0C08- 05 06 07 08 0A （显示修改结果）

3. 移动一段内存区域的内容

格式：*地址1<地址2.地址3M↵

该命令功能为：将地址2到地址3内存区域中的内容移动到从地址1开始的内存区域中。

例6. 将$0300～$0363单元中的内容移动到从$6000单元开始的存储区中。操作如下：

* 6000<300.363M ↙

4. 比较两段内存区域中的内容

格式：*第二段内存首地址<第一段内存首地址.第一段内存末地址V↵

该命令把第一段内存区域中的数据块与第二段内存中相同长度的数据块相比较，考查结果是否一致。如果一致，则退出该命令，屏幕上出现提示符*；否则就显示出第一段地址中内容与第二段内容不一致的地址码及其内容，并在括号内显示第二段对应地址单元中的内容。

例7. 比较$0C00～$0C0C区域的内容与$6000～

$ 600C区域的内容是否相同。$ 0C00～$ 0C0C的内容如前所述；$ 6000～$ 600C的内容设依次为00 01 02 ED 0C 08 03 04 05 06 07 08 0A。操作如下：

```
* C00<6000.600CV ↙        两段内容比较
  6004-0C (FC)            显示表明比较结果不一致
```

在 $ 6004单元中的内容是 $ 0C，在对应单元 $ 0C04中的内容是 $ FC。

5. 检查和修改寄存器内容

格式：* ^E↙

执行此命令屏幕上依次显示出A，X，Y，P，S寄存器内容。

例8. 检查和修改当前CPU各寄存器的内容。

```
* ^E ↙                          检查当前各寄存器内容。
  A=0B   X=EF   Y=00            P=B0   S=F8
*: B1 B2 F8 C0 EF ↙             依次修改各寄存器内容。
* ^E ↙                          检查修改结果。
  A=B1   X=B2   Y=F8   P=C0   S=EF
*                               退出 ^E命令
```

6. 运行机器语言程序的命令

① 连续执行命令

格式1：* 地址G↙

该命令功能为：从命令中指定的地址开始连续执行程序直到执行完一条BRK指令后中断。

例9. 运行一个显示字母A-Z的机器语言程序。程序如下：

机器语言程序				源程序	
0300	A9	C1		LDA	# $ C1
0302	20	ED	FD	JSR	$ FDED

```
0305  18           CLC
0306  69  01       ADC  #$01
0308  C9  DB       CMP  #$DB
030A  D0  F6       BNE  $0302
030C  00           BRK
```

输入和运行上述机器语言程序的操作过程如下：

```
* 300：A9 C1 20 ED FD 18 69 01 C9
       DB D0 F6 00 ↙                输入目标程序
* 300G ↙                            连续运行目标程序
  ABCDEFGHIJKLMNOPQRSTUVWXYZ        显示结果
*
```

格式2：* ^Y↵

该命令强迫监控程序跳到$03F8单元去执行指令。因此，用户应先在$03F8～$03FA单元存放一条转移指令JMP，执行此指令转向用户程序的入口地址，执行用户程序。

② 单步执行命令

格式：* 地址S↵

执行指定地址为首地址的一条机器指令，显示出执行结果时各寄存器的状态，同时还将机器指令反汇编成符号指令，并显示出来。如果还要继续单步执行，可键入S，直到不需要单步执行为止。所以、用单步执行命令来检查、调试程序十分方便。

例10. 将例9中的程序单步执行。

```
* 300S ↙                             执行第一条指令
  0300-  A9  C1     LDA  #$C1        显示执行结果
  A=C1  X=00  Y=00  P=B0  S=F4
* S ↙                                继续单步执行
  0302-  20  ED  FD  JSR  $FDED
```

```
  A=C1      X=00  Y=00  P=0B   S=F4
* S ↙                          继续单步执行
  FDED- 6C  30  00    JMP  ($0036)
  A=C1  X=00  Y=00  P=B0  S=F4
*                              若不继续，则退出S命令
```

③ 跟踪执行命令

格式：＊地址T↲

从指定地址开始跟踪执行机器指令。每执行完一条指令，就显示出该指令存放的地址、机器码和它的反汇编助记符（即符号指令），以及当前各寄存器的内容。显示格式与S命令相同，但T命令是一条一条地自动执行，直到执行BRK 指令中断为止。

7．反汇编命令

格式1：＊地址L↲

功能：把从指定地址开始的20条机器指令翻译成（反汇编成）符号指令，并显示在屏幕上。

例11．反汇编例9中的机器语言程序。

```
* 300L ↙       以下是屏幕上显示的结果（共20行）
  0300-  A9   C1         LDA   #$C1              （第1行）
  0302-  20   ED   FD    JSR   $FDED
  0305-  18              CLC
  0306-  69   01         ADC   #$0.
  0308-  C9   DB         CMP   #$DB
  030A-  D0   F6         BNE   $0302
  030C-  00              BRK
          ⋮               ⋮
  0313-  00              BRK                     （第20行）
  （地址）（机器指令）    （符号指令）
```

格式2：*L↲

该命令是将从当前地址开始的20条机器指令反汇编成符号指令并显示出来。

8．屏幕显示方式命令

① *I↲ 置屏幕显示方式为反相显示（白底黑字）。

② *N↲ 置屏幕显示方式为正常显示（黑底白字）。

9．选择输入/输出设备命令

① 选择输出设备命令

格式：*槽号^P↲ 将输出控制信号传给槽号所指定的连接槽上的接口卡。槽号选择为0～7。比如：

*1^P↲ 启动打印机(打印机接槽号1),以后屏幕上的所有显示均打印出来。

*0^P↲ 停止打印，回到屏幕显示（屏幕的槽号为0)。

*6^P↲ 转入磁盘驱动器。

*3^P↲ 进入中文状态。

② 选择输入设备命令

格式：槽号^K↲ 将输入控制信号传给槽号所指定的连接槽上的接口卡。槽号选择为0～7。比如：

*0^K↲ 进入以键盘为输入设备的状态。

10．磁带输入/输出命令

① 磁带输入命令

格式：地址1．地址2R↲ 将磁带上的数据读入命令指定的内存区域（从地址1到地址2)，磁带上数据的长度必须与指定的存储区长度相等。

② 磁带输出命令

＊地址1．地址2W↙ 将命令指定的内存区间的内容写到磁带上。

11．十六进制数的加、减运算命令

① 加法命令

格式：＊数据1＋数据2↙ 实现两个两位十六进制数相加。比如：

```
* 10+20↙
  =30                    （屏幕上显示的结果。）
*
```

② 减法命令

格式：＊数据1－数据2↙ 实现两个两位十六进制数相减。比如：

```
* 3-4 ↙
  =FF                    （屏幕上显示的结果。）
*
```

12．多重命令

CEC-Ⅰ机的监控程序允许在同一命令行写入多个监控命令（命令之间用空格分隔），但一个命令行的总字符数不得超过254个。如：

＊300L␣300G↙ 前一个命令是把从＄0300单元开始的机器语言程序反汇编成源程序；后一个命令是连续执行从＄0300单元开始的机器语言程序。下面是执行此二重命令后在屏幕上显示的结果。

```
0300-  A9  C1          LDA   #$C1
0302-  20  ED  FD      JSR   $FDED
0305-  18              CLC
```

```
0306-  69  01          ADC  #$01
0308-  C9  DB          CMP  #$DB
030A-  D0  F6          BNE  $0302
030C-  60              RTS
030D-  00              BRK
  ⋮                     ⋮
0313-  00              BRK
ABCDEFGHIJKLMNOPQRSTUVWXYZ
*
```

6.4 监控系统中的通用子程序

CEC-Ⅰ机中有许多可供用户调用的子程序。我们在表6.3中列出了一些常用的子程序，指出了它们的名称，入口地址，操作功能，入口参数（子程序要求在它的主程序中预置哪些寄存器），出口参数（子程序运算结果存放在哪里），以及子程序在执行过程中使用和变更了哪些寄存器等。这样，用户可以不必了解这些子程序的内部结构，就可以很方便地调用这些子程序，减少用户编程时的麻烦。

表中所列"选定的输入设备"一般为键盘；"选定的输出设备"一般为显示器，在BELL中为扬声器。

表6.3　CEC-I机监控程序中的常用子程序

入口地址（十六进制）	名　称	功　　　能	调用前需预置的寄存器	被改变的寄存器
F800	PLOT	在屏幕上A行Y列位置画出一点	A中置行号，Y中置列号	A
F819	HLINE	在屏幕上从A行左边Y列到右边H2列划一条水平线	A中置行号，Y中置起始列号，H2（$C2单元）中置终止列号	A，Y
F828	VLINE	在屏幕上Y列从上边A行到下边V2行划一条竖直线	Y中置列号，A中置竖线的起始行号，V2中置竖线的终止行号	A
F864	SETCOL	设定低分辨率图形的颜色	在A中右半字节设置彩色码	A
F85F	NXTCOL	将现有颜色的彩色码加3，设定低分辨率图形的新颜色		A
FB1E	PREAD	读游戏控制器上第（X）号模拟量输入，结果存于Y中		A，Y
FF3A	BELL	输出一个响铃字符到被选择的输出设备中	A = $87(BEL)	A
FBD9	BELL1	喇叭产生1KHz声响，持续响1/10秒	A = $87	A，Y

续表

入口地址（十六进制）	名　称	功　　能	调用前需预置的寄存器	被改变的寄存器
FBE4	BELL2	喇叭产生1KHz声响，持续时间由Y定	Y中预置持续时间	A，Y
FDED	COUT	输出A中字符到被选择的输出设备	字符的ASCII码在A中	
FDF0	COUT1	输出A中字符到显示器	字符的ASCII码在A中	
FC62	CR	送一个回车换行符到显示器	A＝$8D（CR的ASCII码）	A，X，Y
FD8E	CROVT	送一个回车符到被选择的输出设备	A＝$8D	A
F940	PRNTYX	将X，Y内容以四位十六进制ASCII码送被选择的输出设备	Y＝高字节 X＝低字节	A
F941	PRNTAX	将A，X内容以四位十六进制ASCII码送被选择的输出设备	A＝高字节 X＝低字节	A
F944	PRNTX	将X内容以ASCII码形式送到被选设备	X	A
FDDA	PRBYTE	将A中内容以ASCII码形式送到选定设备	A	A
FDE3	PRHEX	将A中低半字节内容用ASCII码形式输出	A	A

续表

入口地址（十六进制）	名　称	功　　能	调用前需预置的寄存器	被改变的寄存器
F948	PRBLNK	送三个空格到被选输出设备	A = $A0(SP) X = 0	A X
F94A	PRBL2	输出X个空格	A = $A0(SP) X = 空格数目 X = 0表示256个	A X
FDOC	RDKEY	由选定输入设备输入一个字符		Y A = 输入字符
FD1B	KEYIN	读键盘		A = 输入字符
FD6A	GETIN	由选定输入设备输入一行字符(≤256个)	$33单元中放入提示符	A，Y X = 字符个数
FD67	GETINZ	先输出一个回车符，然后转入GETIN	$33中放提示符	同GETIN
FD6F	GETINI	不显示提示符，其它同GETIN		同上
FCA8	WAIT	延时。延时时间为$(13+13.5A+2.5A^2)*1.023\mu s$	A = 延时值	A = 0
FF4A	SAVE	将CPU的寄存器A，X,Y,P,S依次存入零页的$45～$49单元		A,X
FF3F	RESTORE	将零页$45～$49单元中的内容送回到A,X，Y,P,S		A,X，Y,P，S

附录　6502 指令系统表

符号指令	操　作	机器码（16进制）	N	T	标志寄存器P N V · B D I Z C	寻址方式
ADC #$n	A+$n+C→A	69 n	2	2	N V · · · · Z C	立即寻址
ADC $$n_2n_1$	A+($$n_2n_1$)+C→A	6D n_2 n_1	3	4	N V · · · · Z C	绝对寻址
ADC $n	A+($n)+C→A	65 n	2	3	N V · · · · Z C	零页寻址
ADC ($n,X)	A+(有效地址)+C→A	61 n	2	6	N V · · · · Z C	先变址间接
ADC ($n),Y	A+(有效地址)+C→A	71 n	2	5	N V · · · · Z C	后变址间接
ADC $n,X	A+($n+X)+C→A	75 n	2	4	N V · · · · Z C	零页X变址
ADC $$n_2n_1$,X	A+($$n_2n_1$+X)+C→A	7D n_1 n_2	3	4①	N V · · · · Z C	绝对X变址
ADC $$n_2n_1$,Y	A+($$n_2n_1$+X)+C→A	79 n_1 n_2	3	4①	N V · · · · Z C	绝对Y变址
AND #$n	A∧$n→A	29 n	2	2	N · · · · · Z ·	立即寻址
AND $$n_2n_1$	A∧($$n_2n_1$)→A	2D n_1 n_2	3	4①	N · · · · · Z ·	绝对寻址

续表

符号指令	操作	机器码(16进制)	N	T	标志寄存器P N V · B D I Z C	寻址方式
AND $n	A∧($n)→A	25 n	2	3	N · · · · · Z ·	零页寻址
AND ($n,X)	A∧(有效地址)→A	21 n	2	6	N · · · · · Z ·	先变址间接
AND ($n),Y	A∧(有效地址)→A	31 n	2	5	N · · · · · Z ·	后变址间接
AND $n,X	A∧($n+X)→A	35 n	2	4	N · · · · · Z ·	零页X变址
AND $$n_2n_1$$,X	A∧($$n_2n_1$$+X)→A	3D n_1 n_2	3	4①	N · · · · · Z ·	绝对X变址
AND $$n_2n_1$$,Y	A∧($$n_2n_1$$+Y)→A	39 n_1 n_2	3	4①	N · · · · · Z ·	绝对Y变址
ASL $$n_2n_1$$	C←D₇←D₀←0 (M)	0E n_1 n_2	3	6	N · · · · · Z C	绝对寻址
ASL $n		06 n	2	5	N · · · · · Z C	零页寻址
ASL		0A	1	2	N · · · · · Z C	累加器寻址
ASL $n,X		16 n	2	6	N · · · · · Z C	零页X变址
ASL $$n_2n_1$$,X		1E n_1 n_2	3	7	N · · · · · Z C	绝对X变址

续表

符号指令	操作	机器码（16进制）	N	T	标志寄存器P N V · B D I Z C	寻址方式
BCC \$n	C=0 分支	90 n	2	3②	· · · · · · · ·	相对寻址
BCS \$n	C=1 分支	B0 n	2	3②	· · · · · · · ·	相对寻址
BEQ \$n	Z=1 分支	F0 n	2	3②	· · · · · · · ·	相对寻址
BNE \$n	Z=0 分支	D0 n	2	3②	· · · · · · · ·	相对寻址
BVC \$n	V=0 分支	50 n	2	3②	· · · · · · · ·	相对寻址
BVS \$n	V=1 分支	70 n	2	3②	· · · · · · · ·	相对寻址
BMI \$n	N=1 分支	30 n	2	3②	· · · · · · · ·	相对寻址
BPL \$n	N=0 分支	10 n	2	3②	· · · · · · · ·	相对寻址
BIT \$n	A∧（\$n）	24 n	2	3	N V · · · · Z ·	零页寻址
BIT $\$n_2n_1$	A∧（$\$n_2n_1$）	2C n_1 n_2	3	4	N V · · · · Z ·	绝对寻址
BRK	中断	00	1	7	· · · 1 · 1 · ·	隐含寻址

续表

符号指令	操　　作	机器码（16进制）	N	T	标志寄存器P N V · B D I Z C	寻址方式
CLC	0→C	18	1	2	• • • • • • • 0	隐含寻址
CLD	0→D	D8	1	2	• • • • 0 • • •	隐含寻址
CLI	0→I	58	1	2	• • • • • 0 • •	隐含寻址
CLV	0→V	B8	1	2	• 0 • • • • • •	隐含寻址
CMP #$n	A－$n	C9 n	2	2	N • • • • • Z C	立即寻址
CMP $$n_2n_1$	A－($$n_2n_1$)	CD n_1 n_2	3	4	N • • • • • Z C	绝对寻址
CMP ($n,X)	A－(有效地址)	C1 n	2	6	N • • • • • Z C	先变址间接
CMP ($n),Y	A－(有效地址)	D1 n	2	5	N • • • • • Z C	后变址间接
CMP $n, X	A－($n＋X)	D5 n	2	4	N • • • • • Z C	零页X变址
CMP $$n_2n_1$,X	A－($$n_2n_1$＋X)	DD n_1 n_2	3	4①	N • • • • • Z C	绝对X变址
CMP $$n_2n_1$,Y	A－($$n_2n_1$＋Y)	D9 n_1 n_2	3	4①	N • • • • • Z C	绝对Y变址
CMP $n	A－($n)	C5 n	2	3	N • • • • • Z C	零页寻址

续表

符号指令	操　作	机器码（16进制）	N	T	标志寄存器P N V · B D I Z C	寻址方式
CPX #$n	X－$n	E0 n	2	2	N · · · · · Z C	立即寻址
CPX $n	X－($n)	E4 n	2	3	N · · · · · Z C	零页寻址
CPX $$n_2n_1$	X－($$n_2n_1$)	EC n_1 n_2	3	4	N · · · · · Z C	绝对寻址
CPY #$n	Y－$n	C0 n	2	2	N · · · · · Z C	立即寻址
CPY $n	Y－($n)	C4 n	2	3	N · · · · · Z C	零页寻址
CPY $$n_2n_1$	Y－($$n_2n_1$)	CC n_1 n_2	3	4	N · · · · · Z C	绝对寻址
DEC $n	($n)－1→($n)	C6 n	2	5	N · · · · · Z ·	零页寻址
DEC $n,X	($n＋X)－1→($n＋X)	D6 n	2	6	N · · · · · Z ·	零页X变址
DEC $$n_2n_1$	($$n_2n_1$)－1→($$n_2n_1$)	CE n_1 n_2	3	6	N · · · · · Z ·	绝对寻址
DEC $$n_2n_1$,X	($$n_2n_1$+X)-1→($$n_2n_1$＋X)	DE n_1 n_2	3	7	N · · · · · Z ·	绝对X变址
DEX	X－1→X	CA	1	2	N · · · · · Z ·	隐含寻址
DEY	Y－1→Y	88	1	2	N · · · · · Z ·	隐含寻址

续表

符号指令	操作	机器码（16进制）	N	T	标志寄存器P N V • B D I Z C	寻址方式
EOR #$n	$A \oplus \$n \rightarrow A$	49 n	2	2	N • • • • • Z •	立即寻址
EOR $n	$A \oplus (\$n) \rightarrow A$	45 n	2	3	N • • • • • Z •	零页寻址
EOR $n,X	$A \oplus (\$n + X) \rightarrow A$	55 n	2	4	N • • • • • Z •	零页X变址
EOR $n₂n₁	$A \oplus (\$n_2n_1) \rightarrow A$	4D n_1 n_2	3	4	N • • • • • Z •	绝对寻址
EOR $n₂n₁,X	$A \oplus (\$n_2n_1 + X) \rightarrow A$	5D n_1 n_2	3	4①	N • • • • • Z •	绝对X变址
EOR $n₂n₁,Y	$A \oplus (\$n_2n_1 + Y) \rightarrow A$	59 n_1 n_2	3	4①	N • • • • • Z •	绝对Y变址
EOR ($n,X)	$A \oplus$(有效地址)$\rightarrow A$	41 n	2	6	N • • • • • Z •	先变址间接
EOR ($n),Y	$A \oplus$(有效地址)$\rightarrow A$	51 n	2	5③	N • • • • • Z •	后变址间接
INC $n	$(\$n) + 1 \rightarrow (\$n)$	E6 n	2	5	N • • • • • Z •	零页寻址
INC $n,X	$(\$n + X) + 1 \rightarrow (\$n + X)$	F6 n	2	6	N • • • • • Z •	零页X变址
INC $n₂n₁	$(\$n_2n_1) + 1 \rightarrow (\$n_2n_1)$	EE n_1 n_2	3	6	N • • • • • Z •	绝对寻址
INC $n₂n₁,X	$(\$n_2n_1 + X) + 1 \rightarrow (\$n_2n_1 + X)$	FE n_1 n_2	3	7	N • • • • Z •	绝对X变址

续表

符号指令	操作	机器码 (16进制)	N	T	标志寄存器P N V · B D I Z C	寻址方式
INX	X+1→X	E8	1	2	N · · · · · Z ·	隐含寻址
INY	Y+1→Y	C8	1	2	N · · · · · Z ·	隐含寻址
JMP $n₂n₁	$n₂n₁→PC	4C n₁ n₂	3	3	· · · · · · · ·	绝对寻址
JMP ($n₂n₁)	($n₂n₁+1, $n₂n₁)→PC	6C n₁ n₂	3	5	· · · · · · · ·	间接寻址
JSR $n₂n₁	$n₂n₁→PC 返回地址入栈	20 n₁ n₂	3	6	· · · · · · · ·	绝对寻址
LDA #$n	$n→A	A9 n	2	2	N · · · · · Z ·	立即寻址
LDA $n	($n)→A	A5 n	2	3	N · · · · · Z ·	零页寻址
LDA $n,X	($n+X)→A	B5 n	2	4	N · · · · · Z ·	零页X变址
LDA $n₂n₁	($n₂n₁)→A	AD n₁ n₂	3	4	N · · · · · Z ·	绝对寻址
LDA $n₂n₁, X	($n₂n₁+X)→A	BD n₁ n₂	3	4①	N · · · · · Z ·	绝对X变址
LDA $n₂n₁,Y	($n₂n₁+Y)→A	B9 n₁ n₂	3	4①	N · · · · · Z ·	绝对Y变址

续表

符号指令	操　　作	机器码（16进制）	N	T	标志寄存器 P N V · B D I Z C	寻址方式
LDA （$n,X）	（有效地址）→A	A1 n	2	6	N · · · · · Z ·	先变址间接
LDA （$n),Y	（有效地址）→A	B1 n	2	5③	N · · · · · Z ·	后变址间接
LDX #$n	$n→X	A2 n	2	2	N · · · · · Z ·	立即寻址
LDX $n	（$n)→X	A6 n	2	3	N · · · · · Z ·	零页寻址
LDX $n,Y	（$n+Y)→X	B6 n	2	4	N · · · · · Z ·	零页Y变址
LDX $$n_2n_1$$	（$$n_2n_1$$)→X	AE n_1 n_2	3	4	N · · · · · Z ·	绝对寻址
LDX $$n_2n_1$$,Y	（$$n_2n_1$$+Y)→X	BE n_1 n_2	3	4①	N · · · · · Z ·	绝对Y变址
LDY #$n	$n→Y	A0 n	2	2	N · · · · · Z ·	立即寻址
LDY $n	（$n)→Y	A4 n	2	3	N · · · · · Z ·	零页寻址
LDY $n,X	（$n+X)→Y	B4 n	2	4	N · · · · · Z ·	零页X变址
LDY $$n_2n_1$$	（$$n_2n_1$$)→Y	AC n_1 n_2	3	4	N · · · · · Z ·	绝对寻址
LDY $$n_2n_1$$, X	（$$n_2n_1$$+X)→Y	BC n_1 n_2	3	4①	N · · · · · Z ·	绝对X变址

续表

符号指令	操　作	机器码 (16进制)	N	T	标志寄存器 P N V · B D I Z C	寻址方式
LSR	$0 \to \lvert \overline{D_7 \to D_0} \rvert \to \boxed{C}$ M	4A	1	2	0 · · · · · Z C	累加器寻址
LSR $n		46 n	2	5	0 · · · · · Z C	零页寻址
LSR $n,X		56 n	2	6	0 · · · · · Z C	零页X变址
LSR $n₂n₁		4E n₁ n₂	3	6	0 · · · · · Z C	绝对寻址
LSR $n₂n₁,X		5E n₁ n₂	3	7	0 · · · · · Z C	绝对X变址
NOP	空 操 作	EA	1	2	· · · · · · · ·	隐含寻址
ORA #$n	A∨$n→A	09 n	2	2	N · · · · · Z ·	立即寻址
ORA $n	A∨($n)→A	05 n	2	3	N · · · · · Z ·	零页寻址
ORA $n, X	A∨($n+X)→A	15 n	2	4	N · · · · · Z ·	零页X变址
ORA $n₂n₁	A∨($n₂n₁)→A	0D n₁ n₂	3	4	N · · · · · Z ·	绝对寻址
ORA $n₂n₁, X	A∨($n₂n₁+X)→A	1D n₁ n₂	3	4①	N · · · · · Z ·	绝对X变址
ORA $n₂n₁, Y	A∨($n₂n₁+Y)→A	19 n₁ n₂	3	4①	N · · · · · Z ·	绝对Y变址

续表

符号指令	操作	机器码（16进制）	N	T	标志寄存器P N V · B D I Z C	寻址方式
ORA（$n，X）	A∨(有效地址)→A	01 n	2	6	N · · · · · Z ·	先变址间接
ORA（$n），Y	A∨(有效地址)→A	11 n	2	5	N · · · · · Z ·	后变址间接
PHA	A→Ms，S－1→S	48	1	3	· · · · · · · ·	隐含寻址
PHP	P→Ms，S－1→S	08	1	3	· · · · · · · ·	隐含寻址
PLA	S＋1→S，Ms→A	68	1	4	N · · · · · Z ·	隐含寻址
PLP	S＋1→S，Ms→P	28	1	4	恢复P原来状态	隐含寻址
ROL	←$\lvert D_7 \leftarrow D_0 \rvert$←$\lvert C \rvert$←（M）	2A	1	2	N · · · · · Z C	累加器寻址
ROL $n		26 n	2	5	N · · · · · Z C	零页寻址
ROL $n，X		36 n	2	6	N · · · · · Z C	零页X变址
ROL $$n_2n_1$		2E n_1 n_2	3	6	N · · · · · Z C	绝对寻址
ROL $$n_2n_1$，X		3E n_1 n_2	3	7	N · · · · · Z C	绝对X变址

续表

符号指令	操　　作	机器码（16进制）	N	T	标志寄存器P N V · B D I Z C	寻址方式
ROR	$\rightarrow \boxed{C} \rightarrow \underbrace{\boxed{D_7 \rightarrow D_0}}_{M} \rightarrow$ （循环）	6A	1	2	N · · · · · Z C	累加器寻址
ROR $n		66 n	2	5	N · · · · · Z C	零页寻址
ROR $n, X		76 n	2	6	N · · · · · Z C	零页X变址
ROR $$n_2n_1$		6E n_1 n_2	3	6	N · · · · · Z C	绝对变址
ROR $$n_2n_1$, X		7E n_1 n_2	3	7	N · · · · · Z C	绝对X变址
RTI	中断返回	40	1	6	恢复P原来的状态	隐含寻址
RTS	子程序返回	60	1	6	· · · · · · · ·	隐含寻址
SBC #$n	A－$n－C→A	E9 n	2	2	N V · · · · Z C	立即寻址
SBC $n	A－($n)－C→A	E5 n	2	3	N V · · · · Z C	零页寻址
SBC $n, X	A－($n＋X)－C→A	F5 n	2	4	N V · · · · Z C	零页X变址
SBC $$n_2n_1$	A－($$n_2n_1$)－C→A	ED n_1 n_2	3	4	N V · · · · Z C	绝对寻址

续表

符号指令	操　　作	机器码 (16进制)	N	T	标志寄存器P N V · B D I Z C	绝址方式
SBC $n₂n₁, X	A－($n₂n₁＋X)－C→A	FD n₁ n₂	3	4[1]	N V · · · · Z C	绝对X变址
SBC $n₂n₁, Y	A－($n₂n₁＋Y)－C→A	F9 n₁ n₂	3	4[1]	N V · · · · Z C	绝对Y变址
SBC ($n, X)	A－(有效地址)－C→A	E1 n	2	6	N V · · · · Z C	先变址间接
SBC ($n), Y	A－(有效地址)－C→A	F1 n	2	5[3]	N V · · · · Z C	后变址间接
SEC	1→C	38	1	2	· · · · · · · 1	隐含寻址
SED	1→D	F8	1	2	· · · · 1 · · ·	隐含寻址
SEI	1→I	78	1	2	· · · · · 1 · ·	隐含寻址
STA $n	A→($n)	85 n	2	3	· · · · · · · ·	零页寻址
STA $n, X	A→($n＋X)	95 n	2	4	· · · · · · · ·	零页X变址
STA $n₂n₁	A→($n₂n₁)	8D n₁ n₂	3	4	· · · · · · · ·	绝对寻址
STA $n₂n₁, X	A→($n₂n₁＋X)	9D n₁ n₂	3	5	· · · · · · · ·	绝对X变址

续表

符号指令	操　作	机器码（16进制）	N	T	标志寄存器 P N V · B D I Z C	寻址方式
STA $\$n_2n_1$, Y	A→($\$n_2n_1+Y$)	99 n_1 n_2	3	5	· · · · · · · ·	绝对Y变址
STA ($n, X)	A→(有效地址)	81 n	2	6	· · · · · · · ·	先变址间接
STA ($n), Y	A→(有效地址)	91 n	2	6	· · · · · · · ·	后变址间接
STX $n	X→($n)	86 n	2	3	· · · · · · · ·	零页寻址
STX $n, Y	X→($n+Y)	96 n	2	4	· · · · · · · ·	零页Y变址
STX $\$n_2n_1$	X→($\$n_2n_1$)	8E n_1 n_2	3	4	· · · · · · · ·	绝对寻址
STY $n	Y→($n)	84 n	2	3	· · · · · · · ·	零页寻址
STY $n, X	Y→($n+X)	94 n	2	4	· · · · · · · ·	零页X变址
STY $\$n_2n_1$	Y→($\$n_2n_1$)	8C n_1 n_2	3	4	· · · · · · · ·	绝对寻址
TAX	A→X	AA	1	2	N · · · · · Z ·	隐含寻址
TAY	A→Y	A8	1	2	N · · · · · Z ·	隐含寻址

续表

符号指令	操　作	机器码（16进制）	N	T	标志寄存器P N V · B D I Z C	寻址方式
TSX	S→X	BA	1	2	N · · · · · Z ·	隐含寻址
TXA	X→A	8A	1	2	N · · · · · Z ·	隐含寻址
TXS	X→S	9A	1	2	· · · · · · · ·	隐含寻址
TYA	Y→A	98	1	2	N · · · · · Z ·	隐含寻址

说明：1. 表中的符号

N——字节数。

T——时钟周期数。

n——代表两位16进制数。

n_1n_2——代表四位16进制数。

2. 表中的注释

① 如果跨页，指令执行时钟周期数为5。

② 如果跨页，指令执行时钟周期数为4；不分支时T＝2。

③ 如果跨页，指令执行时钟周期数为6。



封面设计：阎欢玲

ISBN 7-5053-1052-6/TP · 172　定价：2.50 元

汇编语言程序设计

電子工業出版社